Formal Methods for Java

Lecture 21: Properties and Listeners in JPF

Jochen Hoenicke

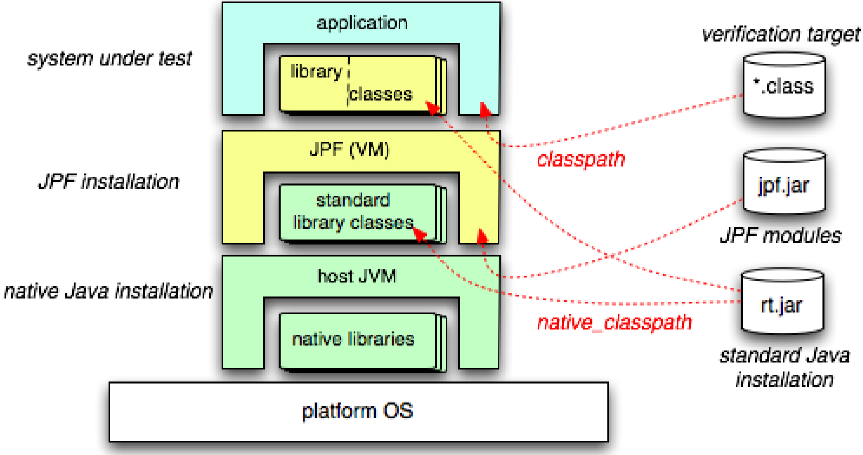
Software Engineering
Albert-Ludwigs-University Freiburg

July 12, 2017

Model checking

- Idea: exhaustively check the system
- Try all possible paths/all possible input values.
- Use search strategies to find errors fast.

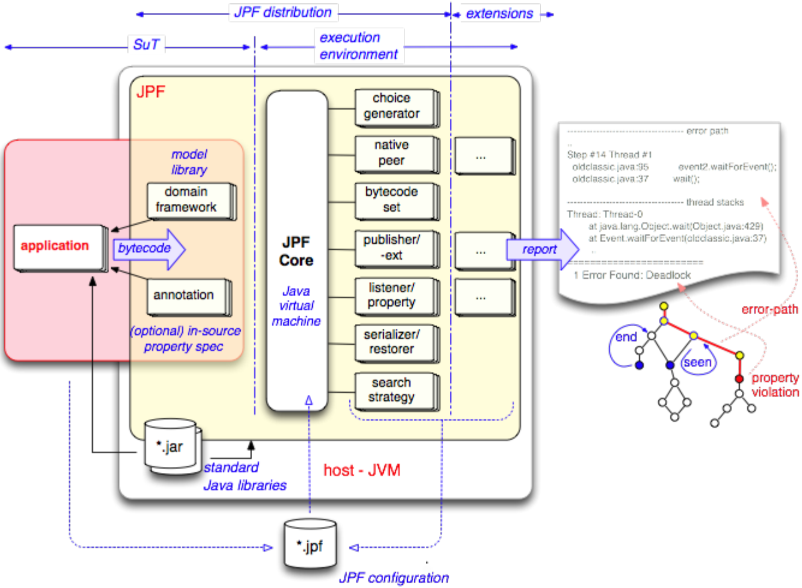
What We Got



http://babelfish.arc.nasa.gov/trac/jpf/wiki

Insights into JPF

JPF Components

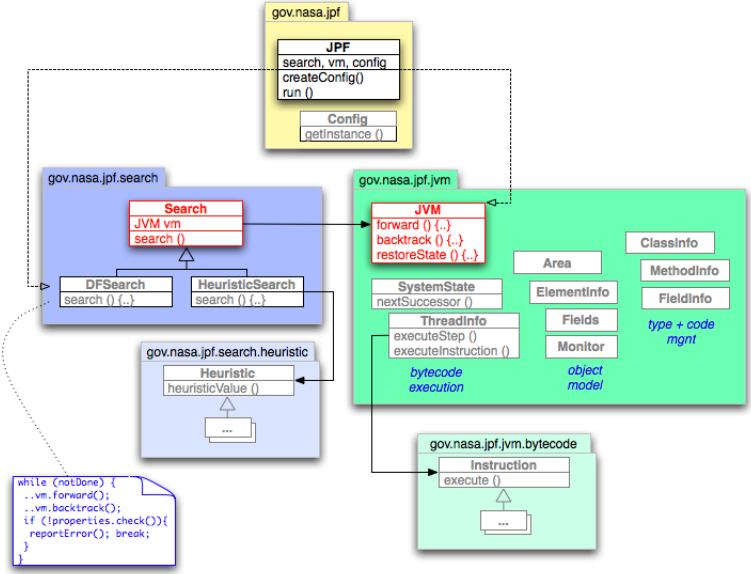


```
---------------------------------- error path
..
Step #14 Thread #1
  oldclassic.java:35          event2.waitForEvent();
  oldclassic.java:37        wait();

---------------------------------- thread stacks
Thread: Thread-0
        at java.lang.Object.wait(Object.java:429)
        at Event.waitForEvent(oldclassic.java:37)
         ..
===========================
  1 Error Found: Deadlock
```

http://babelfish.arc.nasa.gov/trac/jpf/wiki

JPF Core Architecture



```
while (notDone) {
  ..vm.forward();
  ..vm.backtrack();
  if (!properties.check()){
   reportError(); break;
  }
}
```

http://babelfish.arc.nasa.gov/trac/jpf/wiki

Explicit State Model Checking and JPF (1/3)

JVM

Unifies states, produces successor states, backtracking.
Configurations:

vm.class	VM implementation
vm.insn_factory	instruction factory
vm.por	apply partial order reduction
vm.por.sync_detection	detect fields protected by locks
vm.gc	run garbage collection
vm.max_alloc_gc	maximal number of allocations before garbage collection
vm.tree_output	generate output for all explored paths
vm.path_output	generate program trace output
...	and many, many more

Explicit State Model Checking and JPF (2/3)

Search

Selects next state to explore.

Configurations:

search.class	search implementation
search.depth_limit	maximal path length
search.match_depth	only unify if depth for revisit is lower than known depth
search.multiple_errors	do not stop searching at first property violation
search.properties	which properties to check during search
...	further options for each search

Explicit State Model Checking and JPF (3/3)

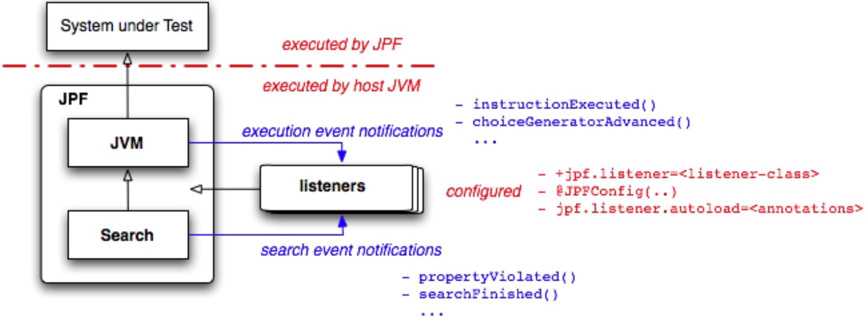
Listener

Evaluate states against properties.

Listeners can influence current transition while properties cannot.

Listener can monitor search and instruction execution.

Own listener can be set with the `listener` configuration option.



http://babelfish.arc.nasa.gov/trac/jpf/wiki

Transition Systems (TS)

Definition (Transition System)

A transition system (TS) is a structure $TS = (Q, Act, \rightarrow)$, where

- Q is a set of states,
- Act a set of actions,
- $\rightarrow \subseteq Q \times Act \times Q$ the transition relation.

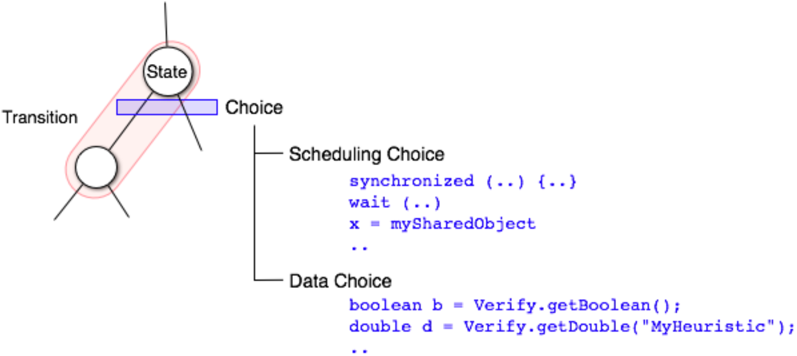
States

Collection of

- thread state (current instruction, stack),
- global variables,
- heap references, and
- trail (path to the state)

Transitions

- Sequence of instructions
- End of transition determined by
  - Multiple successor states (choices)
  - Enforced by listeners (*vm.breakTransition();*)
  - Reached maximal length (configuration `vm.max_transition_length`)
  - End or blocking of current thread



```
synchronized (..) {..}
wait (..)
x = mySharedObject
..
```

```
boolean b = Verify.getBoolean();
double d = Verify.getDouble("MyHeuristic");
..
```

http://babelfish.arc.nasa.gov/trac/jpf/wiki

Choices

Scheduling Choices

Which other thread is runnable?

Partial Order Reduction: Is this thread affected by the current transition?

Controlled by search and VM

Data Choices

Which concrete value to choose for the inputs?

Mostly configured by the user

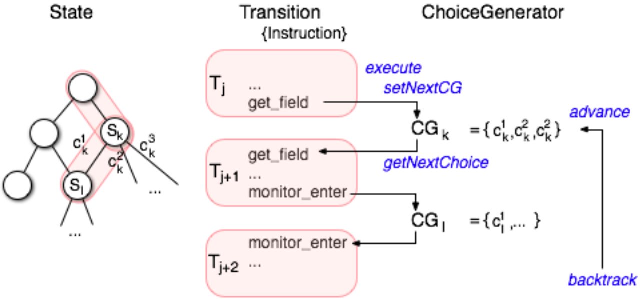
Control Choices

Which branch in the program to take?

Explicit invocation schedule by extensions

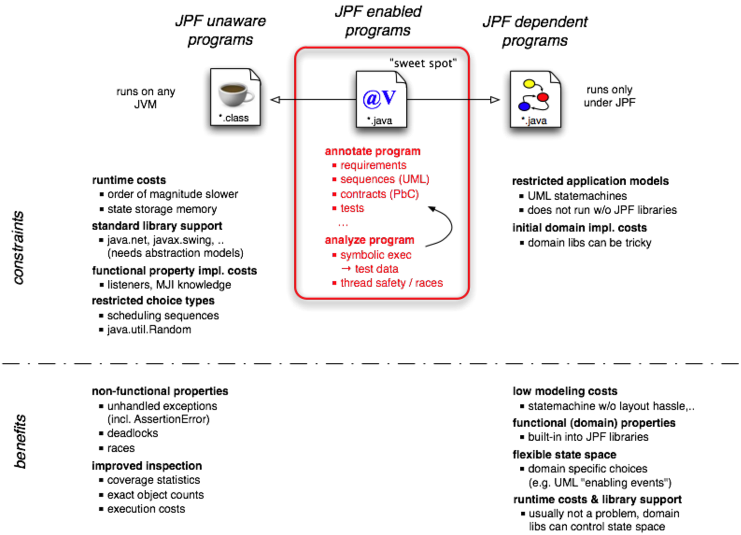
Implementing Choices

- choices encapsulated in ChoiceGenerators (CGs)
- registered by VM, instructions, extensions, or listeners
- `cg.randomize_choices` configures JPF to randomly explore choices



http://babelfish.arc.nasa.gov/trac/jpf/wiki

Applications, JPF, and JPF-Applications



JPF unaware programs

runs on any JVM

JPF enabled programs

JPF dependent programs

runs only under JPF

runtime costs
- order of magnitude slower
- state storage memory

standard library support
- java.net, javax.swing, .. (needs abstraction models)

functional property impl. costs
- listeners, MJI knowledge

restricted choice types
- scheduling sequences
- java.util.Random

annotate program
- requirements
- sequences (UML)
- contracts (PbC)
- tests

...

analyze program
- symbolic exec → test data
- thread safety / races

restricted application models
- UML statemachines
- does not run w/o JPF libraries

initial domain impl. costs
- domain libs can be tricky

non-functional properties
- unhandled exceptions (incl. AssertionError)
- deadlocks
- races

improved inspection
- coverage statistics
- exact object counts
- execution costs

low modeling costs
- statemachine w/o layout hassle,..

functional (domain) properties
- built-in into JPF libraries

flexible state space
- domain specific choices (e.g. UML "enabling events")

runtime costs & library support
- usually not a problem, domain libs can control state space

http://babelfish.arc.nasa.gov/trac/jpf/wiki

Interfering with the Search (1/2)

gov.nasa.jpf.jvm.Verify for choices

getBoolean	Get a Boolean CG
getInt	Get a named integer CG
getIntFromList	Get an integer CG initialized from a list
getObject	Get a named object CG
getDouble	Get a named double CG
getDoubleFromList	Get a double CG initialized from a list
getLongFromList	Get a long CG initialized from a list
getFloatFromList	Get a float CG initialized from a list
random	Get a CG for random values
randomBool	Get a Boolean CG

Interfering with the Search (2/2)

`gov.nasa.jpf.jvm.Verify` for transitions and states

addComment	Add a comment to a state
instrumentPoint	Add a label to a state
atLabel	Check for a label
boring	Hint an uninteresting state
interesting	Conditionally hint an interesting state
ignoreIf	Conditionally prune the search space
beginAtomic	Start an atomic block
endAtomic	End an atomic block
breakTransition	End the current transition

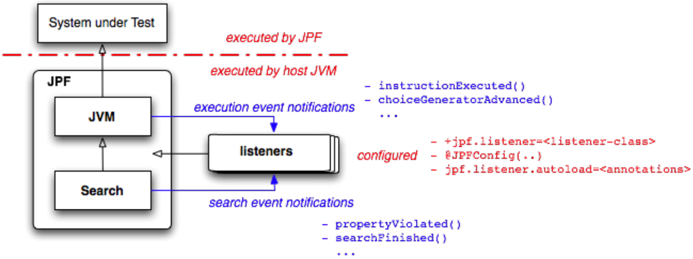
Properties

- Configured with `search.properties`
- Evaluated after every transition
- Base class: *gov.nasa.jpf.Property*
- Properties shipped with JPF Core:
  - *gov.nasa.jpf.jvm.IsEndStateProperty*
  - *gov.nasa.jpf.jvm.NoOutOfMemoryErrorProperty*
  - *gov.nasa.jpf.jvm.NotDeadlockedProperty*
  - *gov.nasa.jpf.jvm.NoUncaughtExceptionsProperty*

Listener

- Configured with `listener` and `listener.autoload`
- Different types:
  - *VMListener* notified about executed instructions, threads state changes, loaded classes, created objects, object monitor events, garbage collections, choice generators, and method enter and exit events
  - *SearchListener* notified about state changes, property violations, and search related events
- Implementation basis for many extensions
- Idea: JPF can check what you can program
- JPF Core comes with many listeners in package *gov.nasa.jpf.listener*

How Listeners Work



http://babelfish.arc.nasa.gov/trac/jpf/wiki

- VM or search notifies listener about next or previous event.
- Listener can act upon this event.
- Listeners can influence VM or search.
- Can annotate objects, fields, operands, and variables with attributes

Writing Our First Listener

Desired Property

A user-specified set of fields and variables should never be assigned to `null`*.*

Chopped into Pieces

- configurable field and variable description
- check for variable and field assignment

JPF Property vs. Listener

- Desired property can be violated by writing a field or variable.
- This does not necessarily break a transition.
- ➥ We need a listener to break the transition and report an error.

Using Utilities (1/2)

gov.nasa.jpf.util.FieldSpec

Utility for specifying field descriptions:

x.y.Foo.bar	field *bar* in class *x.y.Foo*
x.y.Foo+.bar	all *bar* fields in *x.y.Foo* and all its supertypes
x.y.Foo.*	all fields of *x.y.Foo*
*.myData	all fields names *myData*
!x.y.*	all fields of types outside types in package *x.y*

Using Utilities (2/2)

gov.nasa.jpf.util.MethodSpec

Utility for specifying methods:
exact method signature, or:

x.y.Foo.* all methods of class *x.y.Foo*

.(x.y.MyClass) all methods that take exactly one parameter which is of type *x.y.MyClass*

!x.y.*.*(int) no method of any class in package *x.y* or any subpackage that takes exactly one argument that is an `int`

gov.nasa.jpf.util.VarSpec

Utility for specifying local variable descriptions:
Syntax: `MethodSpec`:`VariableName`

Initializing our Listener

```java
public NonNullChecker(Config conf) {
  Set<String> spec = conf.getStringSet("nnc.fields");
  if (spec == null)
    spec = Collections.emptySet();
  nonNullableFields = new FieldSpec[spec.size()];
  int i = -1;
  for (String field : spec)
    nonNullableFields[++i] = FieldSpec.createFieldSpec(field);
  spec = conf.getStringSet("nnc.vars");
  if (spec == null)
    spec = Collections.emptySet();
  nonNullableVars = new VarSpec[spec.size()];
  i = -1;
  for (String var : spec)
    nonNullableVars[++i] = VarSpec.createVarSpec(var);
}
```

Checking the Desired Property Part 1: Fields

Observation

Only two instructions can assign `null` to a field:

- putfield
- putstatic

Basic Idea

If such an instruction wrote to a field we are interested in, check value of that field.

➥ *instructionExecuted* notification

Field Checks

```java
private void checkFieldInsn(FieldInstruction insn) {
  if (isRelevantField(insn)) {
    if (isNullFieldStore(insn)) {
      storeError(vm, insn);
      vm.breakTransition();
    }
  }
}
private boolean isRelevantField(FieldInstruction insn) {
  if (!insn.isReferenceField())
    return false;
  FieldInfo fi = insn.getFieldInfo();
  for (FieldSpec fieldSpec : nonNullableFields) {
    if (fieldSpec.matches(fi)) {
      return true;
    }
  }
  return false;
}
private boolean isNullFieldStore(FieldInstruction insn) {
  FieldInfo fi = insn.getFieldInfo();
  ElementInfo ei = insn.getLastElementInfo();
  return ei.getFieldValueObject(fi.getName()) == null;
}
```

Checking the Desired Property Part 2: Local Variables

Observation

Only one instruction can assign `null` to a local variable:

- astore

We can use our method from before to check that.

Local Variable Checks

```java
private void checkLocalVarInsn(ASTORE insn) {
  if (isRelevantVar(insn)) {
    if (isNullVarStore(insn)) {
      storeError(vm, insn);
      vm.breakTransition();
    }
  }
}
private boolean isRelevantVar(ASTORE insn) {
  int slotIdx = insn.getLocalVariableIndex();
  MethodInfo mi = insn.getMethodInfo();
  int pc = insn.getPosition() + 1;

  for (VarSpec varSpec : nonNullableVars) {
    if (varSpec.getMatchingLocalVarInfo(mi, pc, slotIdx) != null)
      return true;
  }
  return false;
}
private boolean isNullVarStore(ASTORE insn) {
  ThreadInfo ti = vm.getLastThreadInfo();
  int slotIdx = insn.getLocalVariableIndex();
  return ti.getObjectLocal(slotIdx) == null;
}
```

Demo