ORACLE®

Truffle@DSLDI Summer School

DSLDI Summer School 2015/7/17
Christian Humer

VM Research Group, Oracle Labs

Safe Harbor Statement

The following is intended to provide some insight into a line of research in Oracle Labs. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described in connection with any Oracle product or service remains at the sole discretion of Oracle. Any views expressed in this presentation are my own and do not necessarily reflect the views of Oracle.

ORACLE

What to expect

- Overview over Truffle APIs
- How to speculate using Truffle
- Hands-on Demos!

Truffle API

```
class ANode extends Node {

    public int execute() {
        return 21 + 21;
    }

}
```

```
class ARootNode extends RootNode {
    @Child ANode childNode = new ANode();
    @Override
    public Object execute(VirtualFrame arg0) {
        return childNode.execute();
    }
}
```

```
public static void main(String[] args) {
    CallTarget target = Truffle.getRuntime().createCallTarget(new ARootNode());
    target.call();
}
```

Truffle API

```
public interface TruffleRuntime {

    CallTarget createCallTarget(RootNode rootNode);

    DirectCallNode createDirectCallNode(CallTarget target);

    IndirectCallNode createIndirectCallNode();

    Assumption createAssumption();

    <T> T iterateFrames(FrameInstanceVisitor<T> visitor);
    ...
}
```

Truffle API

```
public class CompilerDirectives {

    public static void transferToInterpreter() {...}

    public static void transferToInterpreterAndInvalidate() {...}

    public @interface CompilationFinal {}

    public @interface ValueType {}

    public @interface TruffleBoundary {}

    ...
}
```

Truffle API

Used in the next examples

```
public abstract class Node {
    ...
}


public final class CompilerDirectives {

    public static void transferToInterpreterAndInvalidate() {...}

    public @interface CompilationFinal {}
    ...

}
```

Truffle API Example

```java
class NegateNode extends Node {

    @CompilationFinal boolean minValueVisited;

    public int execute(int operand) {
        if (operand == Integer.MIN_VALUE) {
            if (!minValueVisited) {
                transferToInterpreterAndInvalidate();
                minValueVisited = true;
            }
            return Integer.MAX_VALUE;
        }
        return -operand;
    }
}
```

minValueVisited = true

```java
if (operand == Integer.MIN_VALUE) {
    return Integer.MAX_VALUE;
}
return -operand;
```

minValueVisited = false

```java
if (operand == Integer.MIN_VALUE) {
    transferToInterpreterAndInvalidate();
}
return -operand;
```

Branch Profiles

```
class NegateNode extends Node {

    final BranchProfile minValueProfile = BranchProfile.create();

    public int execute(int operand) {
        if (operand == Integer.MIN_VALUE) {
            minValueProfile.enter();
            return Integer.MAX_VALUE;
        }
        return -operand;
    }
}
```

Condition Profiling

```
class AbsNode extends Node {

    final ConditionProfile smallerZero = ConditionProfile.createBinaryProfile();

    public int execute(int operand) {
        if (smallerZero.profile(operand < 0)) {
            return -operand;
        } else {
            return operand;
        }
    }
}
```

Identity Profiling

```java
public class IdentityValueProfile extends ValueProfile {
    private static final Object UNINITIALIZED = new Object();
    private static final Object GENERIC = new Object();

    @CompilationFinal private Object cachedValue = UNINITIALIZED;

    public <T> T profile(T value) {
        if (cachedValue != GENERIC) {
            if (cachedValue == value) {
                return (T) cachedValue;
            } else {
                transferToInterpreterAndInvalidate();
                if (cachedValue == UNINITIALIZED) {
                    cachedValue = value;
                } else {
                    cachedValue = GENERIC;
                }
            }
        }
        return value;
    }
```

Type Profiling

```
public class ExactClassValueProfile extends ValueProfile {

    @CompilationFinal protected Class<?> cachedClass;
    @Override
    public <T> T profile(T value) {
        if (cachedClass != Object.class) {
            if (cachedClass != null && cachedClass.isInstance(value)) {
                return (T) cachedClass.cast(value);
            } else {
                CompilerDirectives.transferToInterpreterAndInvalidate();
                if (cachedClass == null) {
                    cachedClass = value.getClass();
                } else {
                    cachedClass = Object.class;
                }
            }
        }
        return value;
    }
}
```

Profiles: Summary

- BranchProfiles to speculate on unlikely branches
- ConditionProfile to speculate on binary conditions
- Identity Profiles to speculate on constant values
- Type Profiles to speculate on constant type
- ...

Profiles: Limitations

- Polymorphism:
  - profiles only work with monomorphic situations
  - requires the use of inline caches
- For local speculation only:
  - *`transferToInterpreterAndInvalidate()`* just invalidates the current compilation unit.
  - requires the use of non-local assumptions

Non-local assumptions

```
public interface Assumption {

    boolean isValid();

    void invalidate();
}

Assumption a = Truffle.getRuntime().createAssumption();
```

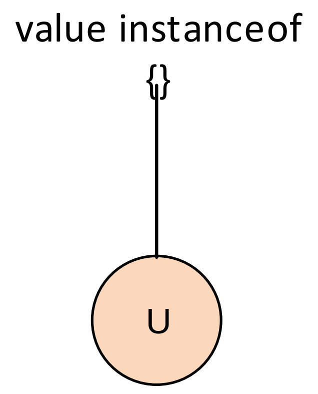
Non-local assumptions

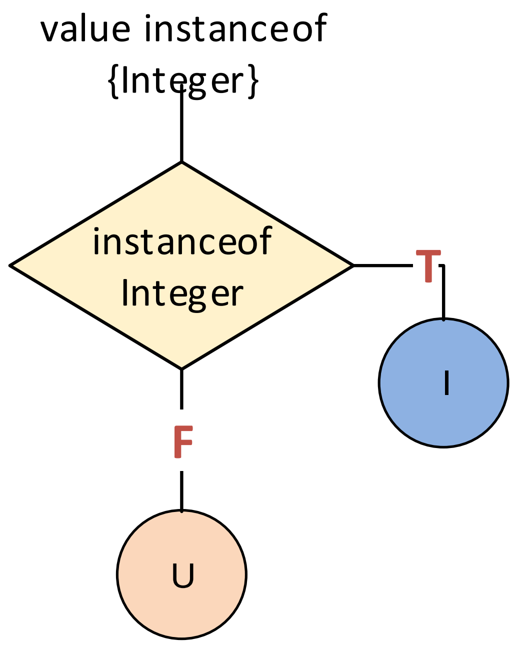
```
public class ANode extends Node {

    private final Assumption assumption = getInstrumentationDisabled();

    public void execute() {
        if (assumption.isValid()) {
            // do nothing
        } else {
            // do instrument
        }
    }
}
```

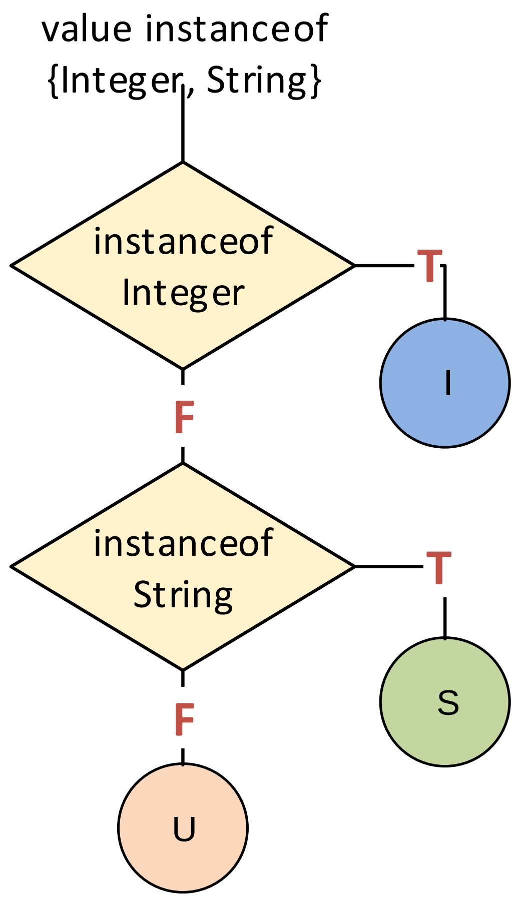
Use-cases for non-local assumptions

- Function redefinition
- Assumed global values
- Tracing / Debugging / Instrumentation
- …

Inline Caching

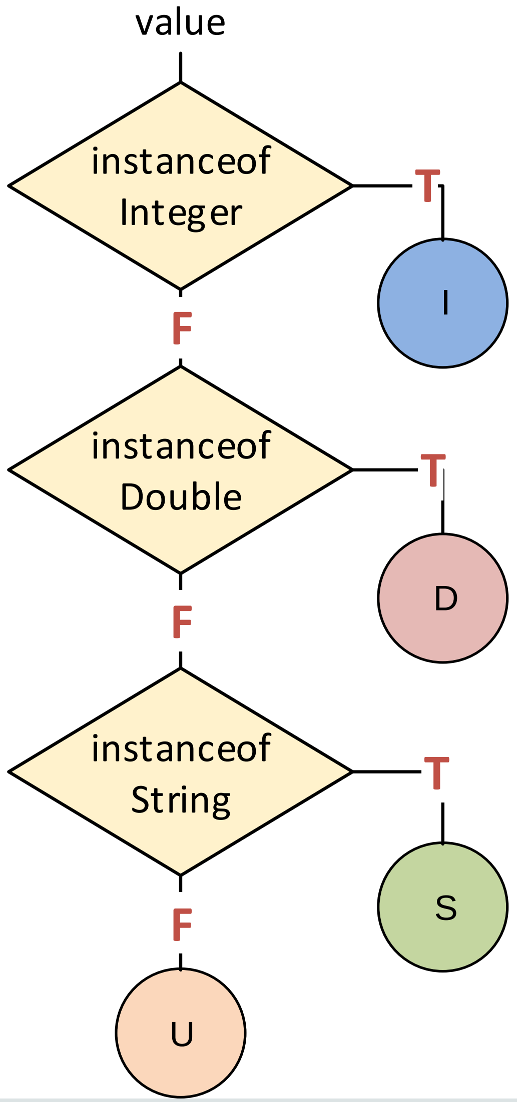






Inline Caching using Truffle DSL

```
class OperationNode extends Node {

    @Specialization
    int doInt(int value) {
        // int implementation
    }

    @Specialization
    double doDouble(double value) {
        // double implementation
    }

    @Specialization
    String doString(String value) {
        // String implementation
    }
}
```



Identity Inline Caching

```
public abstract class ANode extends Node {

    public abstract Object execute(Object operand);

    @Specialization(guards = "operand == cachedOperand", limit = "3")
    protected Object doCached(AType operand,
                    @Cached("operand") AType cachedOperand) {
        // implementation
        return cachedOperand;
    }

    @Specialization(contains = "doCached")
    protected Object doGeneric(AType operand) {
        // implementation
        return operand;
    }
}
```

Type Inline Caching

```
public abstract class ANode extends Node {

    public abstract Object execute(Object operand);

    @Specialization(guards = "operand.getClass() == cachedClass", limit = "3")
    protected Object doCached(AType operand,
                    @Cached("operand.getClass()") Class<? extends AType> cachedClass) {
        AType operand = cachedClass.cast(operand);
        // implementation
        return operand2;
    }

    @Specialization(contains = "doCached")
    protected Object doGeneric(AType operand) {
        // implementation
        return operand;
    }
}
```

Truffle Speculations

Profile, Inline Cache or Assumption?

- Use Profiles where monomorphic speculation is sufficient
- Use Inline Caches for speculations where polymorphism is required
- Use Assumptions for non-local, global speculation

Next up: Simple Language Demos

- SimpleLanguage:
  - Demonstration language for Truffle features (well documented)
- Division speculation
- Zero-overhead tracing

Hardware and Software
Engineered to Work Together

ORACLE®