Truffle based implementation

for the compiler construction exercise 2015

Hermann Schützenhöfer
1226141

Truffle

Java framework to create AST interpreters

Graal

experimental JIT compiler

for the Hotspot VM

Another Framework....?

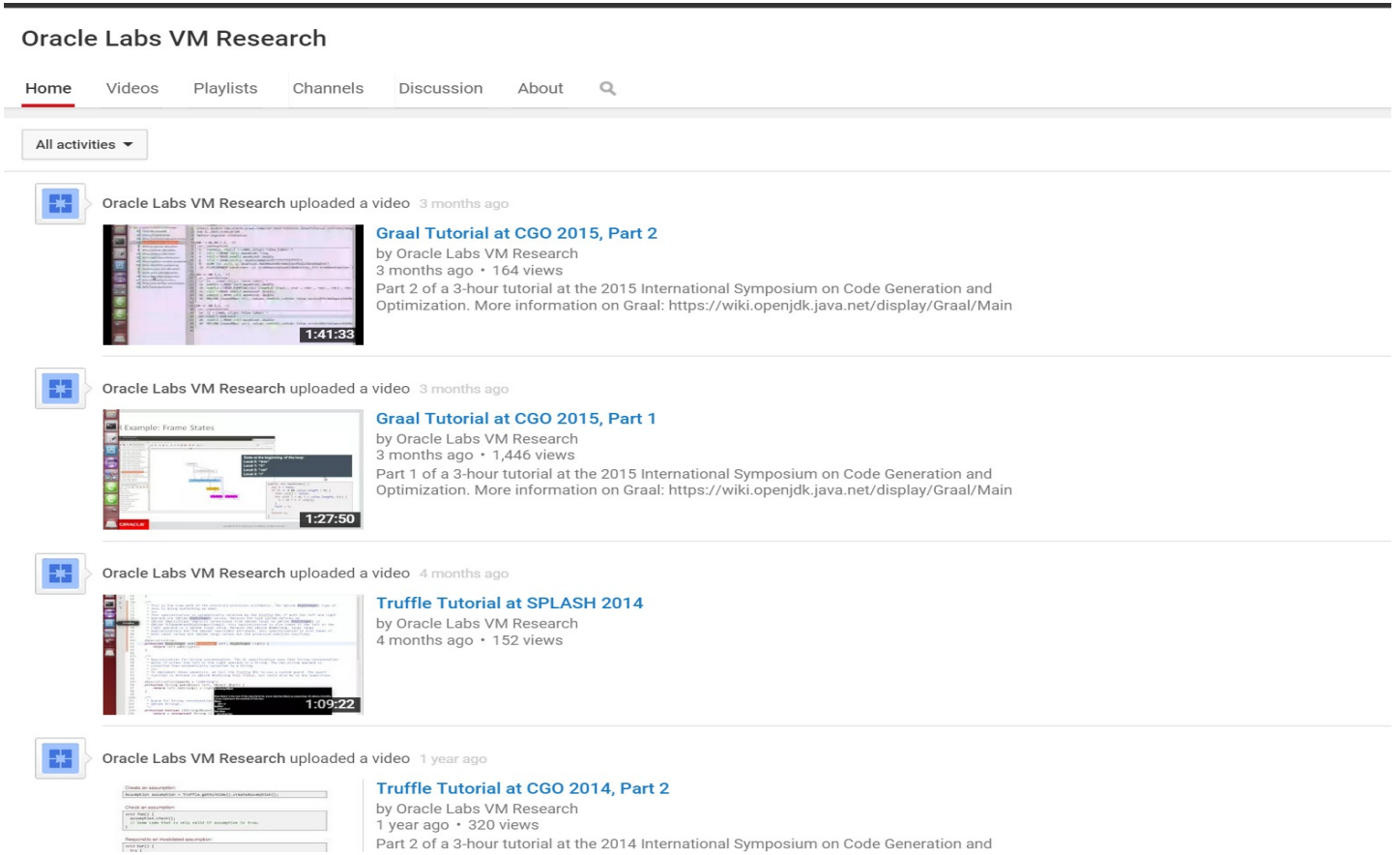
- Runs in a Java VM: JIT, Garbage Collection...
- 'seamless' Java integration
- use Java Tools (IDE, Debugging, ...)
- less development time
- High/competitive performance
  - Speculation
  - Type Specialization
  - Partial Evaluation
  - ...

projects using truffle

- Truffle/JS
- JRuby
- FastR
- ZipPy
- SOM (Smalltalk)
- Truffle/C
- …

Truffle & Graal Introduction

- Youtube: Oracle Labs VM Research



Live Demo

- Add new program features
  - new datatype String
  - String concatenation
- Fibonacci Benchmark
  - Java Code vs. Truffle Interpreter

New datatype String #1

- extend @TypeSystem

```
@TypeSystem({long.class, BigInteger.class,
             String.class, ListType.class, FunctionCall.class})
public class Types {

    @ImplicitCast
    public static BigInteger castBigInteger(long value) {
        return BigInteger.valueOf(value);
    }
}
```

String concatenation #1

- Test program:

```
private final LanguageRunner runner = new LanguageRunner(
        "add = fun x -> fun y -> x + y end end;");
```

- Tests

```
assertEquals("hello world", runner.run("add", "hello ", "world"));
```

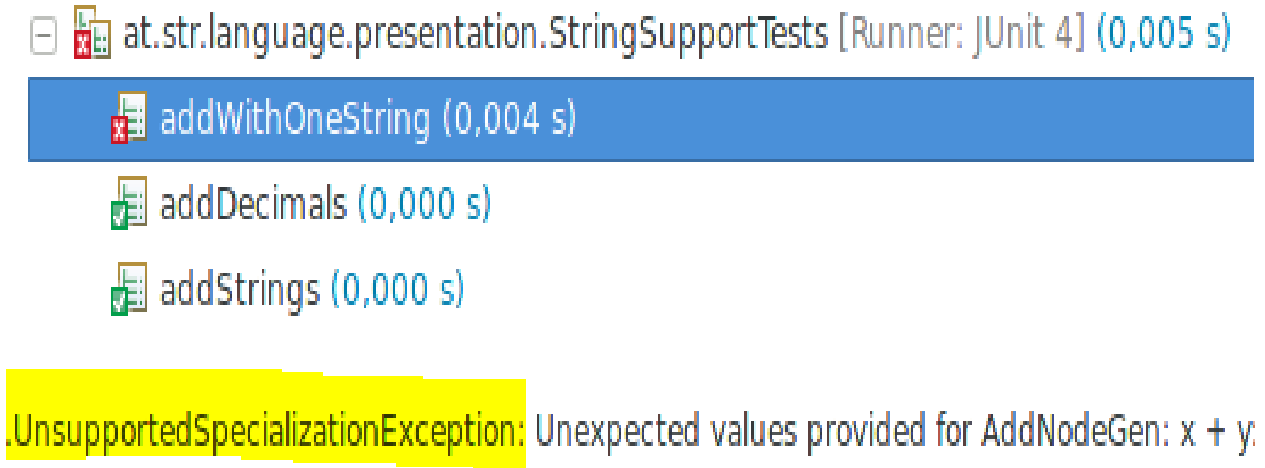
```
assertEquals("hello 2", runner.run("add", "hello ", 2));
assertEquals("3 hello", runner.run("add", 3, " hello"));
```

String concatenation #2

- Implementation

```
@Specialization
protected String add(String left, String right) {
    return left + right;
}
```

- Run tests

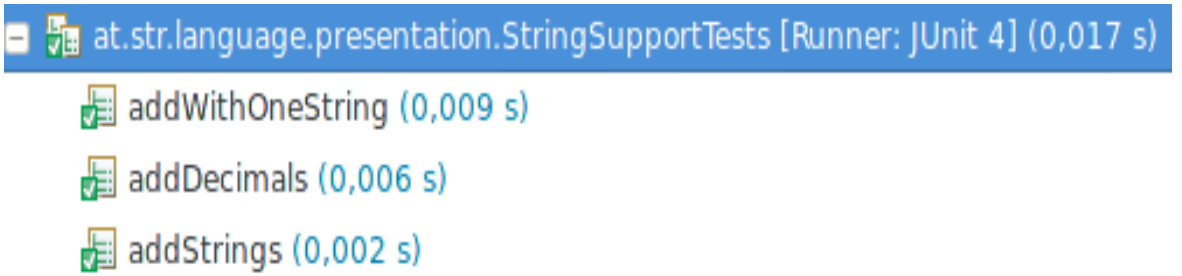


String concatenation #3

- Implementation

```
@Specialization(guards = "isString(left, right)")
protected String add(Object left, Object right) {
    return left.toString() + right.toString();
}

protected boolean isString(Object a, Object b) {
    return a instanceof String || b instanceof String;
}
```

- Run tests



Add new built-in Method #1

- Test program for 'isprime(p)'

```
final LanguageRunner runner = new LanguageRunner(
        "nextPrimeNumber = fun x -> "
                + "if isprime x then x "
                + "else nextPrimeNumber (x+1) "
        +" end end;"
);
```

- Tests:

```
assertEquals(7L, runner.run("nextPrimeNumber", 6));
assertEquals(17L, runner.run("nextPrimeNumber", 15));
assertEquals(127L, runner.run("nextPrimeNumber", 114));
```

Add new built-in Method #2

- Implementation

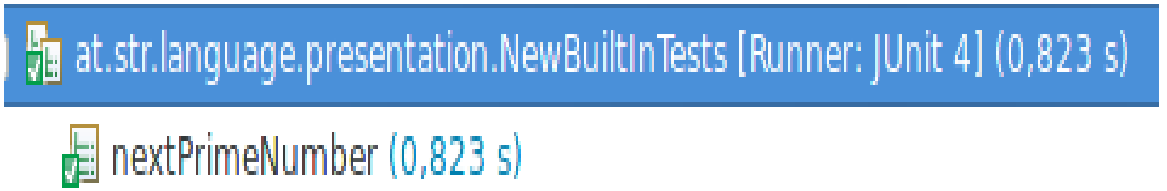
```
@NodeInfo(shortName = "isprime")
public abstract class IsPrimeBuiltin extends BuiltinNode {

    public IsPrimeBuiltin() {
        super(new NullSourceSection("builtin", "isprime"));
    }

    @Specialization
    public long isNum(long value) {
        return isPrime(value) ? TRUE : FALSE;
    }

    @Specialization
    public long isNum(Object value) {
        return FALSE;
    }

    //checks whether an int is prime or not.
    private boolean isPrime(long value) {
```

Add new built-in Method #3

- Register built-in method

```
installBuiltin(IsPrimeBuiltinFactory.getInstance());
```

- Run Tests



Fibonacci Benchmark #1

- Java method

```
private static long fib(long n) {
    if (n < 3) {
        return 1;
    } else {
        return fib(n - 2) + fib(n - 1);
    }
}
```

- Language implementation

```
"fib = fun n -> "
        + " if n < 3 then 1 "
        + "else (fib (n-2)) + (fib (n-1)) "
+ " end end;";
```

Fibonacci Benchmark #2

```
Java Min/Avg/Max:
  84,1ms / 93,2ms / 110,6ms

Language Min/Avg/Max:
  169,2ms / 192,3ms / 239,7ms


Language is in average
x-times slower than java: 2,1x
```