FortressCheck: Automatic Testing for Implicit Parallelism and Generic Properties

Kang Seonghoon
(Joint work with Sukyoung Ryu)

PLRG @ KAIST

August 26, 2010

QuickCheck

...is an automatic random testing tool, originally designed for Haskell.

QuickCheck

...is an automatic random testing tool, originally designed for Haskell. In detail:

"automatic" We don't have to write test cases ourselves; we just have to write descriptive *properties*.

QuickCheck

...is an automatic random testing tool, originally designed for Haskell. In detail:

"automatic" We don't have to write test cases ourselves; we just have to write descriptive *properties*.

"random" QuickCheck generates random test cases using static types.

QuickCheck

...is an automatic random testing tool, originally designed for Haskell. In detail:

"automatic"	We don't have to write test cases ourselves; we just have to write descriptive *properties*.
"random"	QuickCheck generates random test cases using static types.
"testing"	QuickCheck is not a substitute for the proof of correctness; it can be used to show a counter-example of the property, however.

QuickCheck for Other Languages

Due to the conceptual simplicity of QuickCheck, it has been ported to many other languages.

Many implementations depend on the features available to the target languages. For example:

- Those for dynamically typed languages need manual specifications of types.
- Those for object-oriented languages use different ways to determine the most appropriate generator.

Challenge: Port QuickCheck to Fortress.

Why FortressCheck?

A major difference between Fortress and Haskell.

```
square(x) = do
    println("Calculating " x "^2...")
    x^2
end
run() = println(square(3) + square(4))
```

In this code, *square*(3) and *square*(4) may run in parallel and the output is nondeterministic. In general, any side effects within parallel code may produce an unexpected result.

What's New in FortressCheck?

Unlike Haskell, Fortress provides both subtype polymorphism and parametric polymorphism.

Subtype Polymorphism:

$$\texttt{property}\ \mathit{commutativeAddition} = \forall(x\colon \mathrm{Number}, y\colon \mathrm{Number})\,(x + y = y + x)$$

It is impossible to make a single test generator for Number in general, as new subtypes of Number can be added later.

What's New in FortressCheck?

Unlike Haskell, Fortress provides both subtype polymorphism and parametric polymorphism.

Parametric Polymorphism:

$$\texttt{property}\ \mathit{mapLengthInvariant}_1[\![\mathrm{Key}, \mathrm{Val}]\!] = \\ \forall\big(\mathit{map}\colon \mathrm{Map}[\![\mathrm{Key}, \mathrm{Val}]\!], k\colon \mathrm{Key}, v\colon \mathrm{Val}\big) \\ \big(0 \leq \big|\mathit{map}.\mathit{add}(k, v)\big| - |\mathit{map}| \leq 1\big)$$

It is impossible to test $\mathit{mapLengthInvariant}_1$ for every instantiation of Key and Val.

Testing Polymorphism via Reflection

Reflection gives a simple but general way to test polymorphic properties.

$$\texttt{property}\ \mathit{commutativeAddition} = \\ \forall(x\colon \mathrm{Number}, y\colon \mathrm{Number})\,(x + y = y + x)$$

...can be rewritten to a non-polymorphic property:

$$\texttt{property}\ \mathit{commutativeAddition}' = \\ \forall(\mathit{xtype}\colon \mathrm{Type}, \mathit{ytype}\colon \mathrm{Type}) \\ ((\mathit{xtype}\ \texttt{SUBTYPEOF}\ \mathit{numberType}) \wedge \\ (\mathit{ytype}\ \texttt{SUBTYPEOF}\ \mathit{numberType})) \rightarrow \\ \mathit{commutativeAddition}(\mathit{generate}(\mathit{xtype}), \\ \mathit{generate}(\mathit{ytype}))$$

Testing Polymorphism via Reflection

Reflection gives a simple but general way to test polymorphic properties.

$$\begin{array}{l}\texttt{property } \mathit{mapLengthInvariant}_1[\![\mathrm{Key}, \mathrm{Val}]\!] = \\ \quad \forall(\mathit{map}\colon \mathrm{Map}[\![\mathrm{Key}, \mathrm{Val}]\!], k\colon \mathrm{Key}, v\colon \mathrm{Val}) \\ \quad\quad (0 \leq |\mathit{map}.\mathit{add}(k, v)| - |\mathit{map}| \leq 1)\end{array}$$

...can be similarly rewritten to a non-polymorphic property:

$$\begin{array}{l}\texttt{property } \mathit{mapLengthInvariant}'_1 = \\ \quad \forall(\mathit{keytype}\colon \mathrm{Type}, \mathit{valtype}\colon \mathrm{Type})\,(\texttt{do} \\ \quad\quad \mathit{prop} = \mathit{applyStaticParams}(\mathit{mapLengthInvariant}_1, \\ \quad\quad\quad\quad (\mathit{keytype}, \mathit{valtype})) \\ \quad\quad \mathit{prop}(\mathit{generate}(\mathit{mapType}(\mathit{keytype}, \mathit{valtype})), \\ \quad\quad\quad \mathit{generate}(\mathit{keytype}), \mathit{generate}(\mathit{valtype})) \\ \quad \texttt{end})\end{array}$$

Future Work

- Better support for testing implicit parallelism.
- Concise testing language using the extensible syntax of Fortress.