Gen-O-Fix
Embedded Genetic Improvement Programming via Reflection

Jerry.Swan@cs.stir.ac.uk
Michael.Epitropakis@cs.stir.ac.uk
John.Woodward@cs.stir.ac.uk

October 11, 2013
Outline

- **What**: An Embedded Self-Improving System.
- **Why**: Maintainance dominates software lifecycle cost [5] …
- **How**: Reflective Genetic Improvement Programming in Scala.
Gen-O-Fix System Diagram

Gen-O-Fix

Web Application

Source Code

input

generates

Dynamic adaptation for embedded systems.

Source + Binary

improves Performance

improves Power

improves Memory

improves Integrity
Genetic Improvement Programming (GIP)

GIP can be used to:

- Multi-objective trade-off between non-functional properties [2].
- Optimize/improve functional properties.
Embedding Adaptivity

From http://gow.epsrc.ac.uk/NGBOViewGrant.aspx?GrantRef=EP/J017515/1

[DAASE] places computational search at the heart of the processes and products it creates and embeds adaptivity into both.

Gen-O-Fix is a proof-of-concept of a key part of the DAASE manifesto: creating systems with the facility for embedded self-optimization.
Gen-O-Fix Framework Features

- Operates via the new reflection features of the Scala language.
- Source (to Abstract Syntax Tree) to Source + Binary transformation.
- Can be used within ‘always-running’ programs (embedded systems, webservers etc.).
The Scala Programming Language

- Uniquely OO-functional programming hybrid.
- Statically-typed (traps type-errors at compile-time).
- JVM language - fully interoperable with Java (and other JVM languages e.g. Clojure).
Scala in Industry

- Supports expressive webservice frameworks (‘Hello, Web’ in 6 lines of code).
- Increasingly popular for concurrency support (Twitter core rewritten in Scala).
Scala Reflection: Homoiconicity 1

Code as data, data as code.

```scala
// code to data:
var m = 2; var x = 3; var c = 4
val expr = reify((m * x) + c)
println("AST = " + showRaw(expr.tree))

// output:
AST = Apply(Select(Apply(Select(Select(Iden("m"), "elem"), "$times"), List(Select(Iden("x"), "elem"))), "$plus"), List(Select(Iden("c"), "elem")))
```
Scala Reflection: Homoiconicity 2

```scala
// run AST datatype as code:
println("eval = " + expr.tree.eval())

// output:
eval = 10
```
Scala Reflection 3: Rich ASTs and Pattern Matching

Since AST nodes are first class objects in Scala, we have a lot of declarative information available:

- **Well-formedness** of ASTs can be checked by the type-system.
- We can use Scala’s powerful **pattern matching** facility to operate on specific AST fragments:

```scala
  case Apply( Select( Ident( name : TermName ), t2 : TermName ), args )
  if t2 == newTermName( "apply" ) =>
    // manipulate AST
```

- Makes it easier to do context-aware recombination/mutation and other transformations.
- Can help when addressing scalability issues [5].
Search Based Software Engineering (SBSE)

- SBSE starts with only two key ingredients [3]:
- The choice of the representation of the problem. ‘Software is its own substrate’.
- The definition of the objective function.
Gen-O-Fix Framework-Level Parameters

- ‘As simple as possible’:
- Client source code: url that points to Scala code file containing signature $I \to O$ (e.g. $\text{Double} \to \text{Double}$).
- Client functionality fitness: $f : (I \to O) \to \mathbb{R}$.
- Many fitness measures for non-functional properties (e.g. power-consumption) could be supplied ‘as standard’.
A simple proof-of-concept . . .

- A stock-price predictor for shares in David Bowie
- Achieved via univariate symbolic regression . . .
- of a function extracted from the web-application source code.
Created a proof of concept: embedded symbolic regression.

Have started looking at self-repair e.g. Zune and GCD bugs [6].

Next step: Match more complex AST patterns, more powerful transformation rules.
References 1


References II

- **Mark Harman, Phil McMinn, Jerffeson Teixeira de Souza, and Shin Yoo.**
  Search-based software engineering: Techniques, taxonomy, tutorial.

- **John R. Koza.**

- **Claire Le Goues, Stephanie Forrest, and Westley Weimer.**
  Current challenges in automatic software repair.
References III

Claire Le Goues, ThanhVu Nguyen, Stephanie Forrest, and Westley Weimer.  
Genprog: A generic method for automatic software repair.  

Westley Weimer, Stephanie Forrest, Claire Le Goues, and ThanhVu Nguyen.  
Automatic program repair with evolutionary computation.  

David R. White.  
*Genetic Programming for Low-Resource Systems*.  
phdthesis, University of York, UK, December 2009.