Language Technology and Functional Programming

Andy Gill

Information and Telecommunication Technology Center
The University of Kansas

April 3, 2009
## Brief History

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization of Functional Languages</td>
<td></td>
</tr>
</tbody>
</table>

- **University of Glasgow**

- **IA-64/Itanium Low-Level Optimizations**
- **Java VM Optimizations**
- **Legacy Code Translation**
- **Functional Programming and Technology Transfer**
- **Applied Language Research**
- **Functional Programming Products and Services**
- **Applied Language Research (again)**
# Brief History


| University of Glasgow | Optimization of Functional Languages |

## Compilers and Micro-Architectures 1996-1999

<p>| Hewlett Packard | IA-64/Itanium Low-Level Optimizations 120,000 |
| Metrowerks | Java VM Optimizations 200 |
| Semantic Designs | Legacy Code Translation 10 |</p>
<table>
<thead>
<tr>
<th>Brief History</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ph.D. – Compilers &amp; Functional Programming</strong> 1991-1995</td>
</tr>
<tr>
<td><a href="#">University of Glasgow</a></td>
</tr>
<tr>
<td>Optimization of Functional Languages</td>
</tr>
<tr>
<td><strong>Compilers and Micro-Architectures</strong> 1996-1999</td>
</tr>
<tr>
<td><a href="#">Hewlett Packard</a></td>
</tr>
<tr>
<td>IA-64/Itanium Low-Level Optimizations 120,000</td>
</tr>
<tr>
<td>Java VM Optimizations 200</td>
</tr>
<tr>
<td>Legacy Code Translation 10</td>
</tr>
<tr>
<td><strong>Functional Programming and Technology Transfer</strong> 1999-</td>
</tr>
<tr>
<td><a href="#">OGI School of Science &amp; Engineering</a></td>
</tr>
<tr>
<td>Applied Language Research</td>
</tr>
<tr>
<td>Functional Programming Products and Services 4 to 40</td>
</tr>
<tr>
<td><a href="#">Galois</a></td>
</tr>
<tr>
<td><a href="#">The University of Kansas</a></td>
</tr>
<tr>
<td>Applied Language Research (again)</td>
</tr>
</tbody>
</table>
The Science of Programming Languages: Compilation

Program $\rightarrow$ Compiler $\rightarrow$ Executable

C $\rightarrow$ gcc $\rightarrow$ a.out

Java $\rightarrow$ javac $\rightarrow$ bytecode $\rightarrow$ JIT $\rightarrow$ Haskell

Heavy Lifting $\rightarrow$ Optimizations and Representations $\rightarrow$ Semantics $\rightarrow$ Faster, Smaller, Better Executables
The Science of Programming Languages: Compilation

Program $\rightarrow$ Compiler $\rightarrow$ Executable

Haskell $\rightarrow$ \texttt{ghc} $\rightarrow$ \texttt{a.out}

Java $\rightarrow$ \texttt{javac} $\rightarrow$ bytecode $\rightarrow$ \texttt{JIT}

C $\rightarrow$ \texttt{gcc} $\rightarrow$ \texttt{a.out}
The Science of Programming Languages: Compilation

Program → Compiler → Executable

Haskell → `ghc` → `a.out`

Java → `javac` → `bytecode` → `JIT`

C → `gcc` → `a.out`

Faster, Smaller, Better Executables

Optimizations and Representations

Heavy Lifting

Transformations

Semantics
The Science of Programming Languages: Compilation

Program → Compiler → Executable

Haskell → ghc → a.out

Java → javac → bytecode → JIT

C → gcc → a.out

Faster, Smaller, Better Executables

Optimizations and Representations
The Science of Programming Languages: Compilation

Program → Compiler → Executable

Haskell → ghc → a.out
Java → javac → bytecode → JIT
C → gcc → a.out

Semantics

Optimizations and Representations
Faster, Smaller, Better Executables
The Science of Programming Languages: Compilation

Program → Compiler → Executable

Haskell → ghc → a.out
Java → javac → bytecode → JIT
C → gcc → a.out

Semantics

Optimizations and Representations

Faster, Smaller, Better Executables

Heavy Lifting
The Science of Programming Languages: Compilation

Program → Compiler → Executable

Transformations

Haskell → ghc → a.out

Java → javac → bytecode → JIT

C → gcc → a.out

Semantics

Heavy Lifting

Optimizations and Representations

Faster, Smaller, Better Executables
The Science of Programming Languages: Abstraction

Descriptive

Haskell

Java

C

Prescriptive

Expect more of new abstractions! How can they help a programmer be more descriptive? Can we customize abstractions to specific problem domains?
The Science of Programming Languages: Abstraction

- Expect more of new abstractions!
- How can they help a programmer be more descriptive?
- Can we customize abstractions to specific problem domains?
Embedded Domain Specific Languages (EDSLs) provide new abstractions by using powerful language features, not by extending languages.

**EDSLs**

- share syntax, type system and semantics with host language.
- provide additional semantics via a published interface.
- sometimes are just a library based round a categorical structure.
- sometimes provide hooks to allow other tools to execute the user’s program.
- sometimes combine both a library and externally invokable interface.
EDSL Example: Lava

- Lava is a EDSL written in Haskell developed by Xilinx and Chalmers University of Technology in Sweden.
- Based on $\mu$-FP, a calculus for circuits.
- Expresses structural circuits directly.

```
halfAdder (a,b) = (carry,sum)
  where carry = and2 (a,b)
        sum = xor2 (a,b)
```

- Can also capture physical layout and wiring suggestions.
- Uses the abstractions in Haskell to provide abstractions in Lava.
- Circuits built in Lava can be directly executed.
- Circuits can be compiled into VHDL.
- Circuits can also be compared to other circuits.
Islands of Implementations

Lava Structural Model

Compare

Lava Structural Model

Compare

Lava Structural Model

Compare

Lava Structural Model
Islands of Implementations

Lava Structural Model $\rightarrow$ VHDL

Compare

Lava Structural Model $\rightarrow$ VHDL

Compare

Lava Structural Model $\rightarrow$ VHDL

Translation

Translation

Translation

Higher-level Model $\rightarrow$ VHDL

Low-level (signal-based) Behavior Model $\rightarrow$ VHDL
Islands of Implementations

Higher-level Model $\xleftarrow{\text{Compare}}$ Low-level (signal-based) Behavioral Model $\xrightarrow{\text{Compare}}$ Lava Structural Model $\xrightarrow{\text{Compare}}$ VHDL

Lava Structural Model $\xrightarrow{\text{Compare}}$ VHDL

Lava Structural Model $\xrightarrow{\text{Compare}}$ VHDL

Lava Structural Model $\xrightarrow{\text{Compare}}$ VHDL
Islands of Implementations

Higher-level Model \( \xleftarrow{\text{Compare}} \) (signal-based) Behavioral Model \( \xrightarrow{\text{Compare}} \) Low-level Model

Lava Structural Model \( \xrightarrow{\text{Translation}} \) VHDL

Lava Structural Model \( \xrightarrow{\text{Compare}} \) VHDL

Translation

Compare
Islands of Implementations

Higher-level Model \(\xrightarrow{\text{Compare}}\) Low-level (signal-based) Behavioral Model \(\xrightarrow{\text{Compare}}\) Structural Model \(\xrightarrow{\text{Translation}}\) VHDL

Low-level (signal-based) Behavioral Model \(\xleftarrow{\text{Compare}}\) Higher-level Model

Structural Model \(\xrightarrow{\text{Translation}}\) VHDL

Lava Structural Model \(\xleftarrow{\text{Compare}}\) Lava Structural Model

Lava Structural Model \(\xrightarrow{\text{Translation}}\) VHDL

VHDL \(\xrightarrow{\text{Translation}}\) VHDL

VHDL \(\xrightarrow{\text{Translation}}\) VHDL
At Galois, was PI for a hardware back-end for a cryptographic language, Cryptol.

Cryptol is a functional language with powerful abstractions for arbitrary sized, arbitrary dimensioned vectors of bits.

This allowed for high-level specifications of cryptographic algorithms.

The project targeted VHDL from Cryptol specifications.

The Cryptol compiler used the semantics of $\mu$-FP, and well-understood retiming translations to provide world-class circuits from specifications.

These specifications are strongly stylized.

The interesting problems, like taming allocations, remain unsolved.
Research Program at KU

Translating any model to another model involves changing representation. Three pieces of KU research are investigating ways to change representation.

Worker/Wrapper

The worker/wrapper transformation is a theoretical framework for changing the type of a computation in a systematic way.

It captures exactly the preconditions for performing rewrites, as well as unifying previously unrelated translations.

(joint work with the University of Nottingham)

KURE + HERA

KURE is a small language hosted in Haskell for writing transformations as first-class entities, and is used for writing rewrite engines.

HERA is a language on top of KURE, for rewriting Haskell programs directly.

Reflection Support

Using the same program fragment in different contexts is critical to the value offering of Lava and other languages embedded in Haskell. We are exploring a new style of restricted reflection in Haskell, to expand the scope and applicability of these languages.
These three pieces support the larger objective of generating quality hardware descriptions for telemetry circuits.

Example

Viterbi Algorithm → Low-level Behavioral Model of Viterbi → Lava Model of Viterbi → VHDL

- Benefit of a “power steering” approach to making design decisions.
- Research result: What makes an abstraction pliable?
- Low risk of failure:
  - We can already write in Lava.
  - Even a few degrees of freedom will be useful.