Language Technology and Functional Programming

Andy Gill

Information and Telecommunication Technology Center
The University of Kansas

April 3, 2009

Brief History

Ph.D. – Compilers & Functional Programming 1991-1995

University of Glasgow

Optimization of Functional Languages

Brief History

Ph.D. – Compilers & Functional Programming		1991-1995
University of Glasgow	Optimization of Functional Languages	

Compilers and Micro-Architectures		1996-1999
HEWLETT PACKARD	IA-64/Itanium Low-Level Optimizations	120,000
metrowerks Software Starts Here	Java VM Optimizations	200
SEMANTIC DESIGNS, INC.	Legacy Code Translation	10

KU

Brief History

Ph.D. – Compilers & Functional Programming — 1991-1995

University of Glasgow

Optimization of Functional Languages

Compilers and Micro-Architectures — 1996-1999

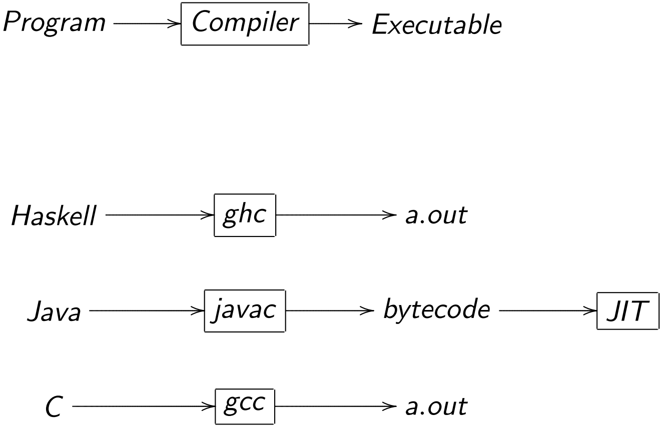
Hewlett Packard	IA-64/Itanium Low-Level Optimizations	120,000
metrowerks Software Starts Here	Java VM Optimizations	200
SD Semantic Designs, Inc.	Legacy Code Translation	10

Functional Programming and Technology Transfer — 1999-

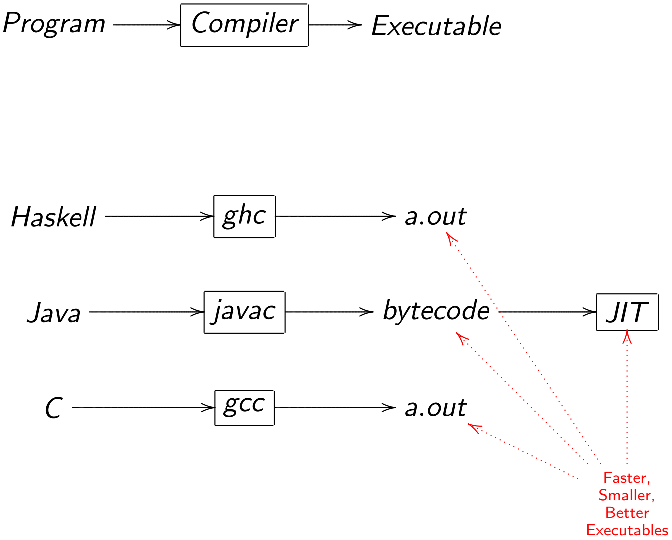
OGI School of Science & Engineering	Applied Language Research	
galois	Functional Programming Products and Services	4 to 40
KU The University of Kansas	Applied Language Research (again)	

The Science of Programming Languages: Compilation

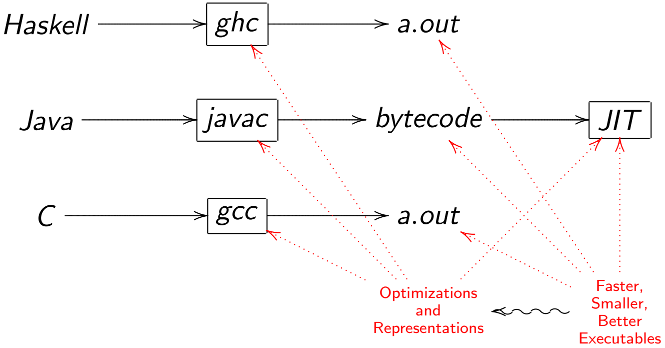
The Science of Programming Languages: Compilation



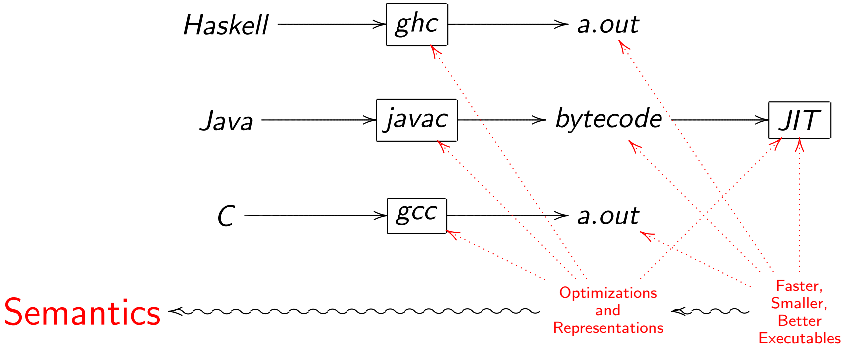
The Science of Programming Languages: Compilation



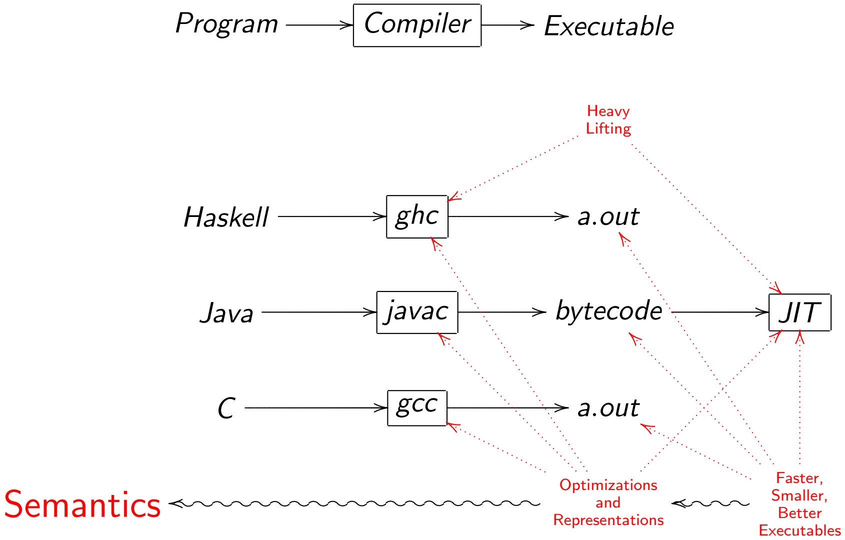
The Science of Programming Languages: Compilation



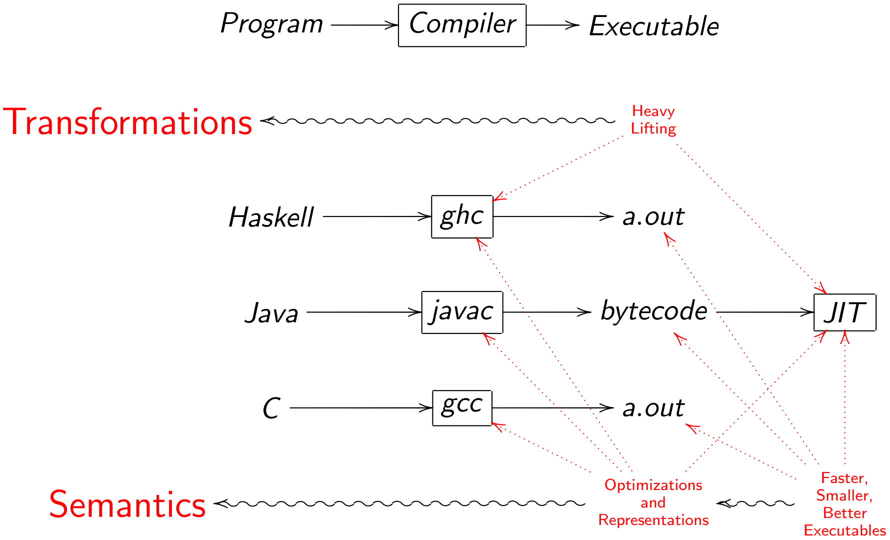
The Science of Programming Languages: Compilation



The Science of Programming Languages: Compilation



The Science of Programming Languages: Compilation



The Science of Programming Languages: Abstraction

Descriptive

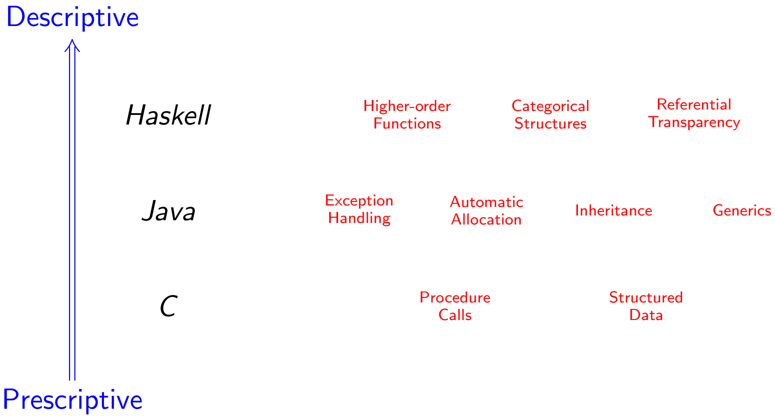
Haskell

Java

C

Prescriptive

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

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.

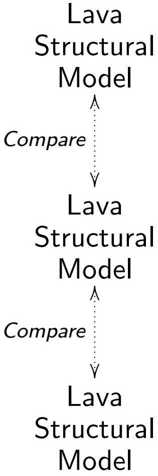
EDSL Example: Lava

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

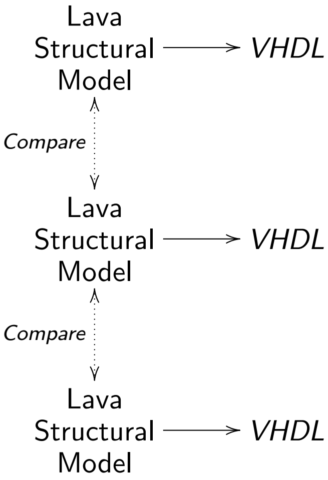
```
halfAdder (a,b) = (carry,sum)
  where carry = and2 (a,b)
        sum   = xor2 (a,b)
```

- 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.

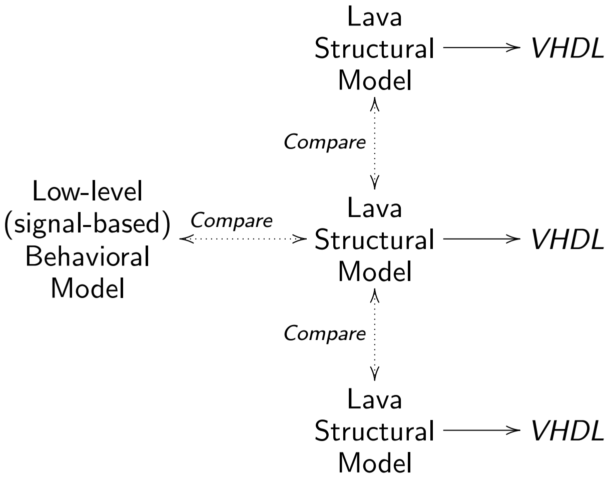
Islands of Implementations



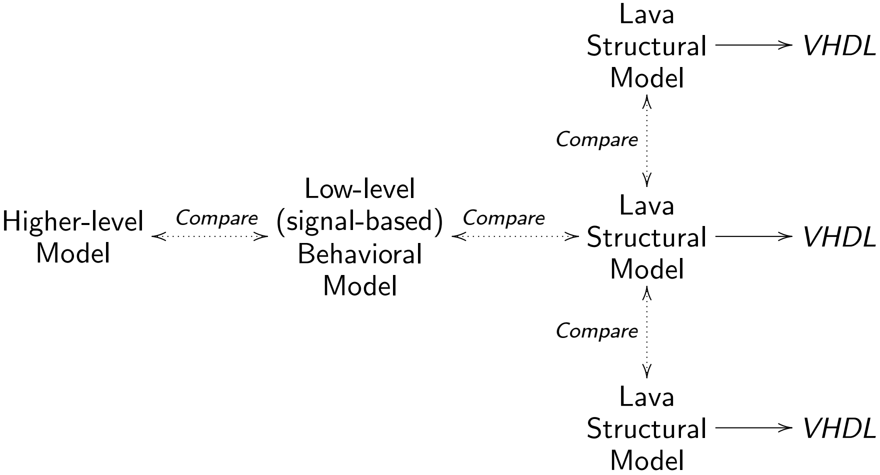
Islands of Implementations



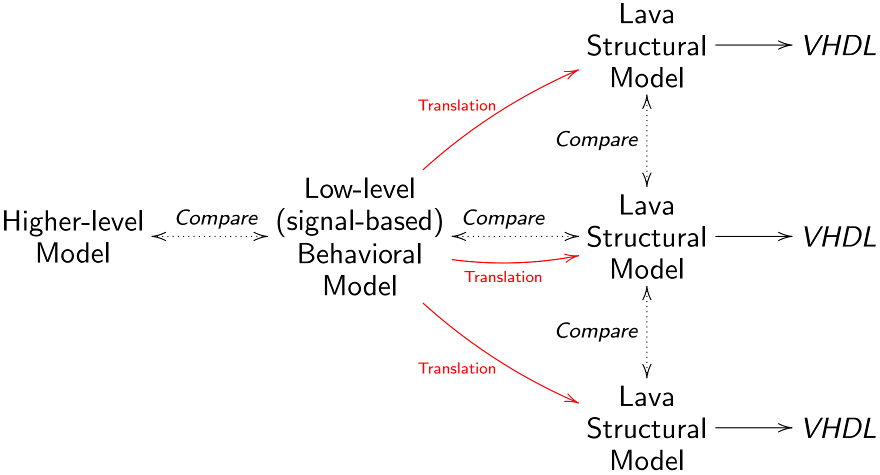
Islands of Implementations



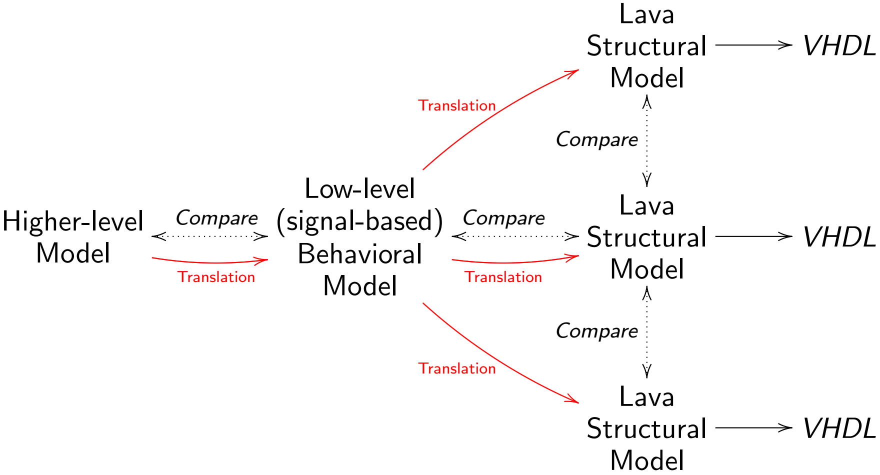
Islands of Implementations



Islands of Implementations



Islands of Implementations



Historical Evidence

- 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 μ-FP, and well-understood retiming translations to provide world-class circuits from specifications.

Research Opportunity

- 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 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.

Research Program at KU (continued)

These three pieces support the larger objective of generating quality hardware descriptions for telemetry circuits.

Example

- 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.

KU