

Abstract Machines

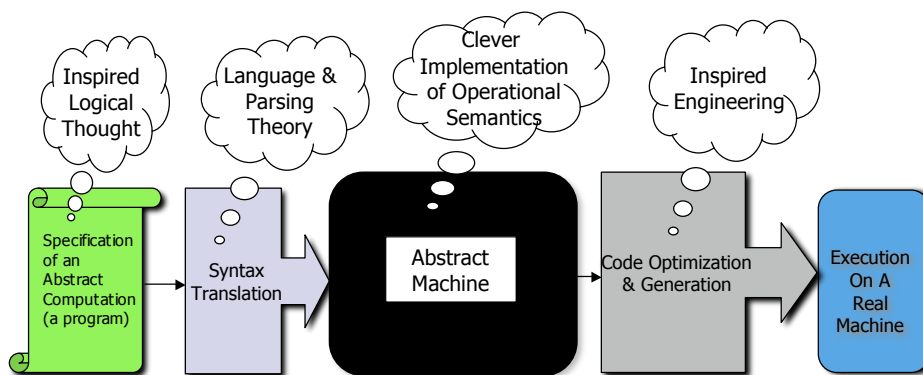
CS395T - Domain Specific Languages

Greg Lavender

Department of Computer Sciences

The University of Texas at Austin

A Black Art?



3/28/05

2

Are There General Design Principles?

- A lot of theory supports parsing/translation
- A lot of practice supports back-end processing
- A lot of experience implementing various kinds of abstract machines
- Are there general design principles for VMs?

3/28/05

3

Abstract Machines

- 40 years of abstract machine development
 - But no general methodology or theory for how to construct them other than "good programming"
 - we know a lot about syntax and parsing
 - i.e., language syntax specification
 - we know a lot about code generation
 - i.e., executable machine specification
 - what about models for all the stuff in-between?
 - type systems
 - optimal evaluations strategies
 - efficient resource management
 - program security
 - error and exception handling
 - Etc.

3/28/05

4

Abstract Machines

- Abstract machines are abstract because they omit many details of real machines
 - bridge the gap between higher-level programming abstractions and the low-level mechanisms of a real machine
 - implement the explicit/implicit operational semantics of a language for writing computations
 - perform a step-by-step execution of a program
 - small-step or big-step
 - may implement mechanisms to support more than one source language (e.g., CLR)
 - manages real machine resources
 - but not necessarily efficiently

3/28/05

5

State of the Art

- Bytecode virtual machines
 - JVM
 - Microsoft Common Language Runtime (CLR)
- Meta-circular interpreters
 - Scheme
- Continuation passing style compiler
 - SML
- Combinator reduction machines
 - Miranda
- Warren Abstract Machine
 - Prolog

3/28/05

6

Imperative & OO Abstract Machines

- Algol Object Code
- P4-machine
- UCSD P-Machine (pascal)
- Basic
- FORTH
- Smalltalk-80
- Self

3/28/05

7

String Processing Abstract Machines

- Snobol4
- Awk
- Nmake (MAM - Make Abstract Machine)
- Tcl
- Perl
- Python

3/28/05

8

Functional Abstract Machines

- Strict evaluation semantics
 - Early Lisp interpreters
 - Landin's SECD machine
 - ISWIM
 - Functional Abstract Machine
 - Categorical Abstract Machine
 - CAML
 - Zinc Abstract Machine
 - Basis for Caml Light, O'Caml and Moscow ML

3/28/05

9

Functional Abstract Machines

- Non-strict (lazy) functional abstract machines
 - SK-machine (a combinator reduction machine)
 - SASL
 - G-machine
 - based on supercombinators
 - Lazy ML, HBC Haskell
 - Krivine machine
 - Three Instruction Machine
 - Supercombinators
 - Spineless-Tagless G-machine
 - Glasgow Haskell Compiler

3/28/05

10

Other Types of Abstract Machines

- Parallel Programming
 - PVM
- Term rewriting
 - ARM, TRAM
- Page description
 - DVI, Postscript, PDF

3/28/05

11

Concrete Abstract Machines

- Burroughs B5000
 - First stack-based hardware architecture
 - Designed original to support Algol execution
- Lisp machines
 - Symbolics
 - TI Explorer
- Pascal Microengine Computer
- ALICE - hardware reduction machine
 - Using a transputer
- Functional
 - Hardware-based G-Machine
 - Burroughs Norma machine
 - Scheme-81 chip
 - Hardware-based WAMs
- microJava, J2ME

3/28/05

12

Core Concepts in FAMs

- A stack represents contexts of nested computations
- An environment maps names to their values
- A closure is used to represent functions as values
- Static and dynamic types
 - polymorphism
- A heap is used to store data
- A garbage collector manages the heap

3/28/05

13

Advanced Concepts for FAMs

- Term reduction based on combinators
- Continuation-passing style
- Monadic style
 - for imperative functional programming w/effects
- Categorical combinators

3/28/05

14

Core Concepts for IAMs?

- Stack-based byte-code execution model
- Secure execution
 - Bytecode verifiers
- Exception handling
- Threading & concurrency
- Networking & communications
- Static vs Dynamic typing
 - Polymorphism
- Efficient execution
 - Just-in-time compilation

3/28/05

15

Coherency?

- Lots of ad hoc techniques
- Is there a way to achieve some operable models that can be effectively reused
- What are the challenges?
- How do we characterize "good" implementations
- Do semantic theories help at all?
- Or is it just engineering (and some hacking?)

3/28/05

16

Monads as a Model?

- We already saw that monads are convenient for writing parsers
 - Can we use them for building the internals of an interpreter?
 - General notion of effects
- Use monads as building blocks
 - they just represent abstract computations
 - compose them together to create building blocks of components of an interpreter
 - requires some understanding of categorical semantics

3/28/05

17

Homework

- Read the following papers:
 - Abstract machines for programming language implementation, by S. Diehl, P. Hartel & P. Sestoft
 - Building Interpreters by Composing Monads, by G. Steele

3/28/05

18