Caffeine – The Habit-Forming Compiler

Justin Bailey

July 15, 2009

What is Caffeine?

An interpreter and x86 compiler for the Habit language.

What is Habit?

Habit is . . .

- Pure
- Functional

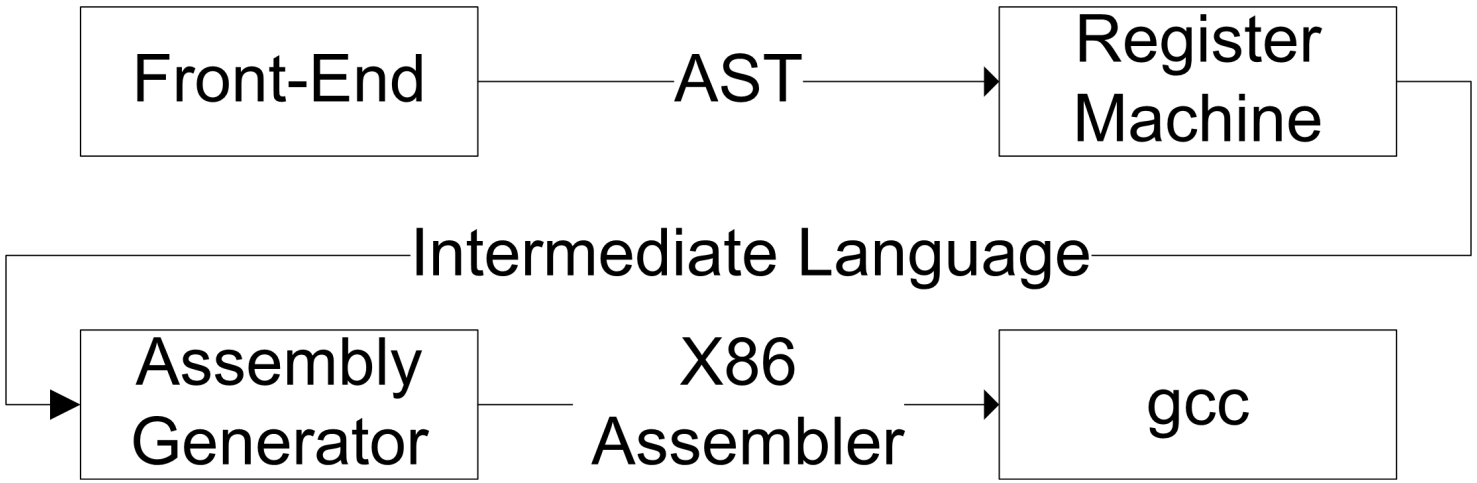
And Caffeine says it is . . .

- Syntactically identical to Haskell
- Strict

What Can Caffeine Do?

- Parameterized algebraic data types
- Higher-order functions, partial application
- Pattern-matching, conditionals, case analysis, guards, etc.
- Global and local definitions
- Mutually recursive definitions (mostly)

Pipeline


Front-End
AST
Register Machine
Intermediate Language
Assembly Generator
X86 Assembler
gcc

Register Machine

- Runs programs in "Intermediate Language" (IL); i.e., an interpreter.
- Simple – easy to map to x86.
- Two special registers – `clo` and `arg`.
- IL used as input to x86 compiler.

x86 Compiler

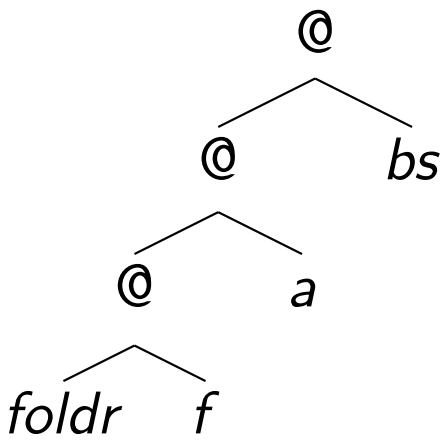
- Produces x86 assembly from IL source.
- Relies on simple runtime for allocation support.
- Produces a result by dumping heap value produced by `main` function.

Closures

Consider

foldr f a bs

The front-end gives this AST:



Closures

Three applications means three functions:

$$foldr\ f\ a\ bs \equiv ((foldr1\ f)\ a)\ bs$$

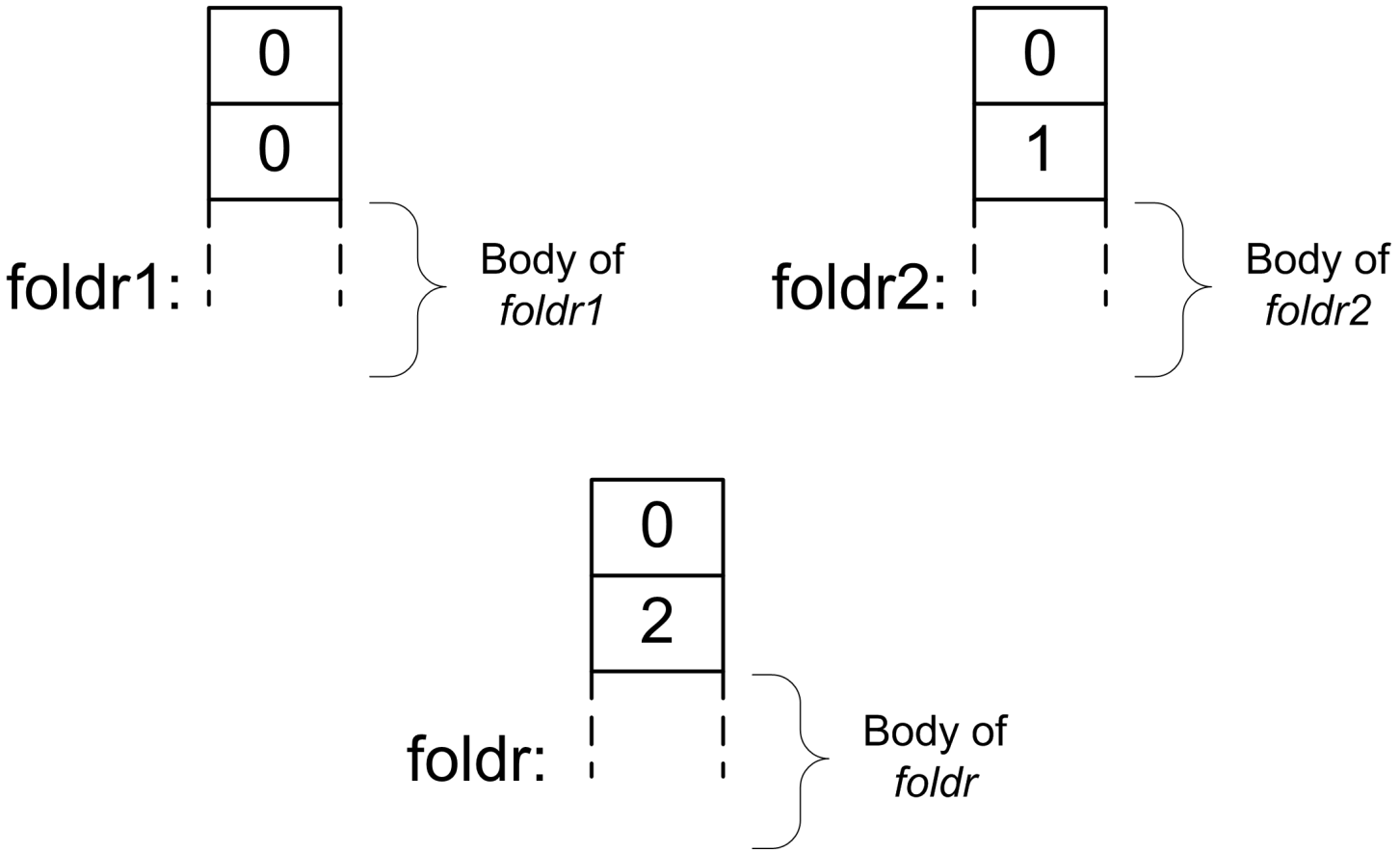
$$foldr1\ f = foldr2\ f$$
$$foldr2\ f\ a = foldr\ f\ a$$
$$foldr\ f\ a\ bs = ...$$

Closures

Register Machine representation:

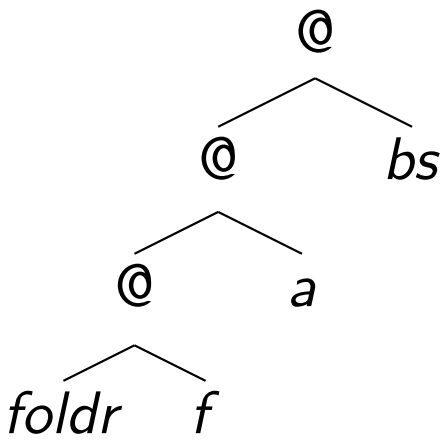
Closures

Assembly representation:



Foldr in IL

foldr f a bs in IL. Recall:



Foldr in IL

Enter implements application. Top-level evaluation:

Enter foldr1 f result0
Enter result0 a result1
Enter result1 bs result2

- *foldr1* – Capture *f*.
- *result0* – Capture *a*.
- *result1* – Evalute *foldr* with *bs*.
- *result2* – Result of evaluation of *foldr f a bs*.

Foldr in IL

foldr1 captures *f*:

AllocC result0 foldr2 1
Store arg $(\mathit{result0}, 0)$ -- arg special.
Ret result0

foldr2 copies *f* and captures *a*:

Load $(\mathit{clo}, 0)$ *f* -- clo special.
AllocC result1 foldr 2
Store f $(\mathit{result1}, 0)$
Store arg $(\mathit{result1}, 1)$
Ret result1

Foldr in IL

foldr is:

$$foldr\ f\ a\ Nil = a$$
$$foldr\ f\ a\ (Cons\ b\ bs) = f\ b\ (foldr\ f\ a\ bs)$$

Check *Nil* first:

$$Load\ (clo, 0)\ f$$
$$Load\ (clo, 1)\ a$$
$$FailT\ arg\ Nil\ lab28$$
$$\ldots$$
$$Ret\ a$$

Foldr in IL

Next, check *Cons*:

lab28 : *FailT arg Cons lab32*

Evaluate *foldr f a bs* and store in *reg37*.

Load $(arg, 0)$ *b*
Load $(arg, 1)$ *bs*
Enter foldr1 f reg35
Enter reg35 a reg36
Enter reg36 bs reg37

Foldr in IL

Finally, evaluate *f a reg37*.

Enter f b reg38
Enter reg38 reg37 result2
...
Ret result2

Foldr in Assembler

Three key instructions from IL to explore

- *Enter* – Application.
- *FailT* – Conditionals/Discrimination.
- *AllocC* – Allocation.

Enter in Assembler

```
# "Enter foldr1 f result0
pushl %esi
pushl %edi
movl foldr1, %edi
movl $f, %esi
call *(%edi)
```

FailT in Assembler

```
    # "FailT arg Nil lab28
    movl (%esi), %ecx
    movl (%ecx), %ecx
    cmpl $0x348, %ecx
    jnz lab28
...
    # "FailT arg Cons lab32
    movl (%esi), %ecx
    movl (%ecx), %ecx
    cmpl $0x3b8, %ecx
    jnz lab32
```

AllocC in Assembler

Recall *foldr1* captures *f* and points to *foldr2*:

AllocC result0 foldr2 1
Store arg $(\textit{result0}, 0)$ -- arg special.
...

In assembler:

```
# AllocC result0 foldr2 1
pushl $foldr2
call _alloc
addl $0x4, %esp
movl %eax, -4(%ebp)
# Store arg (result0,0)
movl %esi, %edx
movl -4(%ebp), %ecx
movl %edx, 4(%ecx)
```

AllocC in Assembler

Preceding `$foldr2` we find the *info table* for the closure:

```
     .long 0x0
     .long 0x1
foldr2:
```

Composability

"Compiler as Service"

Composable Signatures

- Register Compiler: $Module / [Group]$

  $compile :: Supply\ Int \rightarrow Module \rightarrow [Group]$

- Assembly Compiler: $[Group] / Program$[1]

  $assemble :: Supply\ Int \rightarrow [Group] \rightarrow Program$

[1] Dirty secret – $Program \equiv [String]$.

What it Does Not Do

- Primitive types (*Int*, *String*, etc.)
- Typeclasses
- Input/Output
- Multi-module programs
- Garbage Collection

What it Could Do

All of the above ...

REPL

Debugger

Staged compilation

Where to Find Out More

Code available in cg509. The `README` will tell you what to do.

Samples

- `tests\Even.hs`
- `tests\Fib.hs`
- `tests\Foldr2.hs`