Compiladores II

Fabio Mascarenhas – 2018.1

http://www.dcc.ufrj.br/~fabiom/comp2

Máquinas Virtuais

- Uma máquina virtual é uma técnica de implementação de linguagens de programação que é um meio termo entre interpretar diretamente a árvore sintática do programa e compilá-lo para código nativo

- Como uma máquina real, a máquina virtual tem instruções, “registradores” e “memória”, mas essas partes estão mais próximas das operações da linguagem que da máquina real: mais fácil de compilar

- Interpretar um programa de uma máquina virtual é mais eficiente que interpretar diretamente a árvore sintática

Uma VM simples

- VMs podem ser implementadas em qualquer linguagem, mas tipicamente são implementadas em C ou C++ por razões de eficiência, então vamos usar C
- Começaremos com uma máquina virtual para um subconjunto da linguagem *SmallLua* que estávamos usando
- Um programa será apenas uma sequência de comandos, sem definição de funções
- A função `print` será uma primitiva que imprime o resultado de uma expressão

SmallLua sem funções

```
bloco <- stat*
stat  <- "while" exp "do" bloco "end" / "local" "id" "=" exp /
         "id" "=" exp / "if" exp "then" bloco ("else" bloco / '') "end" /
         "print" "(" exp ")"
exp   <- lexp ("or" lexp)*
lexp  <- rexp ("and" rexp)*
rexp  <- cexp (rop cexp)*
cexp  <- aexp ".." cexp / aexp
aexp  <- mexp (aop mexp)*
mexp  <- sexp (mop sexp)*
sexp  <- "-" sexp / "not" sexp / "false" / "true" / "number" /
         "string" / id
rop   <- "<" / "==" / "~="
aop   <- "+" / "-"
mop   <- "*" / "/"
```

Representando valores

- Nossa linguagem tem números, strings e booleanos, mas como os representamos?
- Em uma máquina virtual queremos uma representação *uniforme*, já que as estruturas do interpretador vão precisar armazenar coleções heterogêneas desses valores
- A forma mais fácil é representar cada valor como um ponteiro para seu valor real no heap, chamamos essa representação de *boxing*
- *Boxing* é problemático quando temos operações aritméticas, já que cada operação implica na alocação de um novo valor no heap

Small integers

- Uma otimização comum em máquinas virtuais é aproveitar o fato de que o alocador de memória do sistema sempre aloca endereços múltiplos de 2/4/8, o que permite usar esses bits como uma *tag*

- Tipicamente o último bit 0 indica um ponteiro para algum valor no heap, e o último bit 1 indica que os primeiros 31/63 bits são um número inteiro

- Aritmética com inteiros não precisa mais alocar valores no heap

- Não vamos usar isso em SmallLua, já que nossos números vão ser sempre *doubles*

NaN-boxing

- A representação que os processadores usa para números de ponto flutuante não usa todo o espaço dado pelos 64 bits do tipo double
- Se os primeiros 13 bits (os 13 bits mais significativos) de um double são 1 então ele é um NaN (not-a-number), não importa os valor dos outros 51 bits
- Desses 51 bits, o processador só produz NaNs em que todos eles são 0
- Além disso, ponteiros na arquitetura x64 só usam 48 dos 64 bits disponíveis, então podemos codificar um ponteiro dentro de um NaN, e ainda sobram quase 3 bits para uma tag (quase para garantir que todos os 51 bits não for 0, então temos espaço para 7 tags)

Tagged unions (valores “gordos”)

- NaN-boxing depende de detalhes da arquitetura dos processadores, então não é portável
- Uma terceira representação é representar valores com uma estrutura em que um dos campos é a tag e o outro campo é uma união dos diversos tipos de valor
- Temos uma representação portável que não precisa usar boxing para valores como números inteiros e de ponto flutuante, em troca de consumir mais memória
- É a representação mais simples

Máquinas de pilha vs registradores

- Outra escolha importante é a arquitetura da máquina virtual: pilha vs registradores
- Em uma máquina de pilha as instruções operam implicitamente em uma pilha de operandos: simples de gerar código, instruções menores
- Em uma máquina de registradores, as instruções operam em “registradores virtuais”: instruções maiores, mas o programa precisa de menos instruções, o que aumenta a eficiência
- Cada função ganha seu próprio conjunto de registradores, e esse conjunto pode ter o tamanho que ela precise, então não existe o problema de “alocação de registradores”

SmallLua sem funções - instruções

Despachando instruções

- A maneira mais simples de implementar o laço do interpretador é com um grande switch
- Essa maneira é portável, e o desempenho é bom em processadores modernos
- Em processadores mais antigos uma otimização era usar *gotos computados,* uma extensão do GCC
- Essa segunda maneira é menos portável, e a versão mais eficiente dela, em que as instruções são substituídas por endereços, aumenta o tamanho do programa

Benchmarks

TU + Goto : 0.015625

TU + Switch: 0.015625