A Linguagem Scheme

Daniel Carlos Casarotto, João Gabriel Sapuchay Chiste

Ciências da Computação 4ª fase, 2003
Departamento de Informática e Estatística (INE)
Universidade Federal de Santa Catarina (UFSC), Brasil, 88040-050
Fone (0xx48) 333-9999, Fax (0xx48) 333-9999
casaroto@inf.ufsc.br, jg@inf.ufsc.br

Resumo

Este artigo apresenta uma breve introdução sobre a linguagem funcional Scheme, abordando suas principais características, implementações de Scheme e os princípios básicos da linguagem com alguns exemplos.

Palavras-chave: Scheme, Funcional, Lisp

Abstract

This article presents a brief introduction about the Scheme funcional language, approaching its main characteristics, Scheme implementations and the basics of the language with some examples.

Key-words: Scheme, Funcional, Lisp.

Introdução

Scheme é uma versão estática e recursiva da linguagem de programação Lisp inventada por Guy Lewis Steele Jr. e Gerald Jay Sussman. Ele foi desenvolvido para apresentar uma semântica simples e clara, apresentando poucas formas diferentes de formar expressões. Uma grande variedade de paradigmas de programação são encontrados em Scheme, entre eles, programação funcional, e programação por passagem de mensagens.

Scheme foi uma das primeiras linguagens de programação a incorporar lambda cálculo nos procedimentos da classe principal. Também proporciona a usabilidade de regras de escopo estático e estruturas em bloco em uma linguagem tipada. Scheme foi o primeiro grande dialeto de Lisp a distinguir procedimentos de expressões lambda e símbolos, a usar um único ambiente léxico para todas as variáveis, e a validar a posição do operador de uma chamada de procedimento do mesmo modo de uma posição de operando. Confiando totalmente em chamadas de procedimentos para expressar iteração, Scheme enfatizou que procedimentos recursivos são essencialmente "vá para" que passam argumentos. Scheme foi a primeira linguagem de programação largamente utilizada a adotar procedimentos de escape na classe principal, de qual todos as estruturas de controle previamente conhecidas podem ser sintetizadas. Mais recentemente, construindo sobre o design de aritmética genérica de Common Lisp, Scheme introduziu o conceito de números exatos e inexatos. Scheme também é a primeira linguagem de programação a suportar macros higiênicos, que permitem que a sintaxe de uma linguagem estruturada em blocos possa ser extendida confiavelmente.

MIT Scheme

MIT Scheme roda em muitas plataformas Unix , assim como em Microsoft Windows e IBM OS/2. Possui uma rica biblioteca em tempo real, um poderoso "debugger", um compilador, e um editor parecido com o ambiente Emacs.

Outras Implementações

Guile, GNU's Ubiquitous Intelligent Language for Extension, é uma implementação de Scheme com várias facilidades adicionais convenientes. Foi designado para que você possa associá-lo a uma aplicação ou utilidade para fazê-lo extensível.

Kawa compila Scheme em código Java.

Scheme 48 é uma pequena implementação baseada em interpretador de código de byte. Roda em qualquer máquina com endereçamento de 32-bit que tenha um compilador ANSI C e suporte para POSIX.

Scsh, um Scheme Shell, é uma versão de Scheme para Unix baseado em R4RS Scheme. Roda na maioria das plataformas Unix.

Skij é uma implementação parcial de Scheme implementado e integrado com Java.

STk, é uma versão de Scheme com uma interface Tk.

Números

Os Números são dados simples disponibilizados pela linguagem Scheme, que representam o próprio valor que devolvem. Por exemplo, o número 5, quando integrado numa expressão, devolve o valor 5, como seria de esperar. O número 12E3 é a representação exponencial do valor 12000, pois equivale a 12 x 10^3 (o expoente 3 é o número à direita de E), enquanto que o número 12E-3 representa o valor 0.0035, por ser equivalente a 35 x 10^{-3}.

Interpretador

O interpretador é responsável pela resposta ao utilizador, pois está num ciclo permanente de leitura-cálculo-escrita.

Expressões

Scheme disponibiliza várias funções primitivas, chamadas assim por fazerem parte da definição da linguagem, como os operadores matemáticos + (adição), - (subtração), * (multiplicação), / (divisão).

Em Scheme é usada a notação pré-fixada, exemplos:

```
> (+ 10 25)
35
> (- 10 15 5)
-10
```

Identificadores

Muitas vezes é necessário definir novas entidades dando nomes. Também quando precisamos definir novas funções ou simplesmente criar um identificador. Para isso é necessária uma forma especial de Scheme, a forma "define", que se representa na forma genérica:

```
(define identificador expressão)
```

Exemplo:

```
> (define lado 5)
lado
```

Acabou de ser criado o identificador lado, com o valor 5 associado. Isto pode ser verificado ao pedir o cálculo do seu valor.

```
> lado
5
```

Definindo Funções

É possível criar novas funções. Em vez de ligar um número a um identificador, como se fez em lado, vamos agora associar uma tarefa específica.

Por Exemplo a função dobro, que recebe um valor numérico qualquer e devolve o dobro desse valor:

```
(define dobro    ;definido o nome da função
(lambda (x)      ;x é o único parâmetro.
(* 2 x)))        ;aqui a função é definida
```

Na definição desta função, o valor que deve ser dobrado pode ser qualquer valor numérico, ele foi então representado simbolicamente por x. Diz-se que x é um parâmetro da função, neste caso, o único parâmetro da função.

A chamada da função quadrado faz-se colocando entre parênteses o nome da função, seguido do argumento.

```
> (dobro 3)
6
```

Na chamada (dobro 3), o valor 3 é o argumento. Este valor será associado ao parâmetro x, sendo devolvido (* 2 3) igual a 6, pois a definição de dobro é (* 2 x).

```
> (dobro (+ 3 2))
10
```

Nesta chamada, primeiro é calculado o valor de (+ 3 2), que é 5, e depois é calculado o dobro.

Forma genérica de uma função:

```
(define nome-da-função
(lambda (parametro-1 parametro-2 ...)
corpo-da- função ))
```

Estruturas de Seleção

As entidades com valores #t e #f são designadas de booleanas e as expressões de que resultam booleanos são designadas de predicados. Por exemplo, (< 30 40) é um predicado com valor #t, pois 30 e menor que 40, e (< 30 20) e um predicado com valor #f, pois 30 não é menor que 20.
Outra forma do Scheme, if:

(if expressão-predicado
expressão-consequente
expressão-alternativa)

A regra de cálculo da forma especial if é a seguinte:
Calcular a expressão-predicado;
Se resultar #t, é calculada a expressão-conseqüente;
Se resultar #f, é calculada a expressão-alternativa.
As funções de relação disponibilizadas pelo Scheme são: > (maior), < (menor), = (igual), >= (maior ou igual), e <= (menor ou igual).

Exemplos:
> (if (> 5 12) 8 9)
9
>(if (<= 5 1) 3 4)
4

As funções lógicas disponibilizadas pelo Scheme são: and (e), or (ou), not (não).
Faz parte da cultura Scheme terminar com ? o nome das funções que são predicados, ou seja, que devolvem valores booleanos. Algumas que o Scheme disponibiliza são: negative?, positive?, zero?, even?, odd?, boolean?, symbol?, procedure?, number?, integer?, real? e outros.

Em certas ocasiões é preferível usar a forma especial cond:

```
 (cond (predicado-1 exp1-1 exp1-2 ...)  ;cláusula 1
(predicado-2 exp2-1 exp2-2 ...)          ;cláusula 2
...
(else exp-else-1 exp-else-2 ...) )       ;cláusula else
```

A regra de cálculo da forma especial cond á a seguinte:
Calcular os predicados, começando por predicado-1, até encontrar um predicado com valor #t; logo que se encontre um predicado #t, calcular as expressões correspondentes. Se não encontrar qualquer predicado com valor #t, calcular as expressões correspondentes a else.

A cláusula else é opcional. Se não existir e o cálculo de todos os predicados resultar #f, nenhuma das expressões de cond será calculada.

Exemplo da função Fatorial:

```
(define fat                  ;nome da função
  (lambda (n)                ;parâmetro n
    (if (= n 0)              ;se n=0
       1                     ;retorna 1
        (* n (fat (- n 1)))))) ;senão retorna fat(n-1)
```

Conclusão

Com esse artigo podemos ter uma noção da linguagem Scheme, uma implementação do paradigma funcional, ela é muito poderosa, apresenta uma semântica simples e clara, muito próxima do cálculo lâmbda. Possui várias implementações, o que mostra o interesse nessa linguagem. Possui implementações multi-plataforma e integradas com Java. Isso tudo torna Scheme uma boa opção dentre as inúmeras linguagens funcionais.

Referências

[1] The Scheme Programming Language
http://www.swiss.ai.mit.edu/projects/scheme/

[2] F. Nunes Ferreira, *Introdução à programação com Scheme*, 1999
http://www.fe.up.pt/~jlopes/teach/2001-02/IP1/livro/