Интерпретатор на Scheme

Катерина Георгиева, спец. “Компютърни науки”

16.01.2022

—

Курсов проект по Функционално програмиране

—

ФМИ, СУ “Св. Климент Охридски”
зимен семестър 2021/2022г.

Увод

Проектът се фокусира върху реализация на интерпретатор за езика Scheme, позволяващ интерпретиране и изпълняване на код. Реализиран е на езика Racket. Основната идея зад решението е да интерпретира код, който е подаден под формата на списък с помощта на функцията evall, оценява даден текст, подаден под формата на списък до някакъв функционален обект.

Кратък преглед на структурата на проекта

Двете основни функции за този проект са interpret и evall.
Функцията interpret получава кода за интерпретиране под формата на списък и отделните изрази за оценяване се отделят като отделни елементи в списъка:
`(interpret '[(<израз1>) (<израз2>) ... (<израз$_n$>)]`
Тази функция прилага evall над всеки израз от списъка, за да може да се оцени самостоятелно.
Логиката на evall се състои в това да проверява дали подадения израз може да се сведе до един от следните изрази:

1. Примитивен израз, чиято оценка е самият израз
2. Предварително зададена операция или променлива, чиято оценка се съдържа в настоящата среда
3. Условен If-Else оператор
4. Оператор Cond
5. Define форма
6. Lambda функция

Ако не отговаря на условията за нито един от тези изрази, връща съобщение за грешка от вида на raise-user-error.
Проверките (3), (4) , (5) и (6) са на един и същи принцип, така че обяснението ще бъде подробно само за проверка (3), останалите проверки са аналогични.

Проверка (1) проверява дали изразът, който сме подали, е от тип boolean, number, char, string или procedure, ако е така, връща самият израз.
Проверка (2) проверява дали изразът ни е променлива или предварително зададена операция, като търси дали този израз е вече дефиниран в глобалната ни среда. Изразите, които са зададени предварително са `+, -, *, /, abs, <, <= , >, >=, =, cons, list, car, cdr, cdr, #t, true, #f, false,` както и `null?` и `number?`. Те се съдържат в основната среда E, която представлява хеш таблица, чийто ключ е цитат от израз `('<израз>)`, а стойността е оценката на самия израз.

Ако търсеният израз не се съдържа в основната среда E, получаваме съобщение за грешка.
Проверка (3) проверява дали изразът ни if-else оператор, като проверява първия елемент от този израз дали е if. Ако е така, изразът се оценява със съответната функция, която parse-ва булевия израз, изразът, който се оценява, ако булевият израз е истина и изразът, който трябва да се изпълни в противен случай, тъй като до този момент те не са оценени. Тъй като вече тези изрази, така взети са във формат на S-expression ги оценява с помощта на функция, чиято цел е да оценява S-expression изрази.
Проверка (4) проверява дали изразът ни е cond на същия принцип, по който се проверява и за проверка (3) и ако изразът е cond, се изпълнява функция, която го оценява. Тъй като изразът ни е списък от списъци, проверяваме вътрешните списъци на основния. Те са двойки и са на един и същи принцип, така че винаги (освен в последния случай) имаме двойка от типа `(<булев израз>, <израз>)`
Оценяваме първия израз и ако той връща #t, оценяваме и втория израз, в противен случай преминаваме към следващата двойка от списъка. Дъното на тази рекурсивна проверка е ако елементът ни е последен, в противен случай просто оценяваме втория израз директно и връщаме неговата оценка.
Проверка (5) проверява дали даден израз е define израз по аналогичен начин на останалите и към моментната версия на интерпретатора, оценява израза, ако неговата оценка добавя нова променлива към глобалната среда. Предстои и да се напише процедура, която да може да оценява функции и да връща обект от тип procedure.
Проверка (6) проверява дали изразът ни е lambda израз с подадена променлива или примитивен израз за оценка на целия lambda израз. Раздробява израза на 3 локални променливи в let, които са за параметрите на израза, тялото на израза и примитивните изрази или променливи, които са подадени за оценка на lambda израза. Предстои и да се напише процедура, която да може да оценява функции и да връща обект от тип procedure.

Използвани източници за прилагане на материал, предаден извън лекциите по Функционално програмиране:
Хеш структура