Mac0316/Mac5754

Exercicio Programa 1
(Entrega: 16/09/2024, 7:00hs)
Prof. Alan M.Durham

Neste EP iremos apenas fazer incrementos pequenos no interpretador visto em classe. O objetivo é entender o funcionamento do Racket e do nosso interpretador. No interpretador disponível no Edisciplinas, implementamos as operações de multiplicação e soma como operações primitivas e subtração como açúcar sintático. Aumentamos a linguagem do interpretador que agora inclui:

1. Números
2. Booleanos implementados como números (0 é falso, qualquer outro valor verdadeiro)
3. +, -, *
4. <
5. if
6. And
7. booleanos.

Porém, estes devem ser implementados utilizando números, para que o interpretador possa retornar apenas um tipo. Neste ep vocês devem aumentar o número de operações disponíveis para incluir:

1. **Menos unário (“~”)**
2. Divisão (“/“)
3. Or (“or")
4. Igualdade (“= “)
5. Menor (“<“)
6. Not (“not”)
7. **Maior (“>”)**
8. If (“if”)

NOTA: AS OPÇÕES EM VERMELHO DESTACADAS DEVEM SER IMPLEMENTADAS COMO AÇÚCAR SINTÁTICO

CUIDADO: Implementem “~" (menos unário) como multiplicação

DETALHES ADMINISTRATIVOS:

- Este EP é INDIVIDUAL. Programas com evidencias de cola terão a nota ZERADA (dica: ao trocarem idéias, não troquem código, apenas descrevem em português duvidas e soluções)
- O prazo de entrega termina as 7:00 da manhã (para não interferir com horário de aulas).
- Penalidade por atraso: 20% da nota por dia corrido. (Ou seja em 5 dias o EP não vale mais nada).
- **Voce deve entregar um arquivo .rkt com seu interpretador e os testes que realizou. Tente colocar comentários breves no seu EP para facilitar a compreensão.**

Exemplo de funcionamento de uma solução feita por mim. N#fotem que, embora tenhamos as constantes #t e #f, o if e as funções booleanas trabalham com números.

```
> (interpS (parse '(+ 5 (~ 3))))
- number
2
> (interpS (parse '(if (= (+ (~ 5) 7) (+ 1 1)) (+ 8 2) (/ 4 2))))
- number
10
> (interpS (parse '#t))
- number
1
> (interpS (parse '#f))
- number
0
> (interpS (parse '(= 5 6)))
- number
0
> (interpS (parse '(and (< 5 6) (> 10 9))))
- number
0
> (interpS (parse '(not (= 5 6))))
- number
1
> (parse '(not (= 5 6)))
- ArithS
(notS (equalS (numS 5) (numS 6)))
> (parse '(- 5 6))
- ArithS
(bminusS (numS 5) (numS 6))
> (desugar (parse '(- 5 6)))
- ArithC
(plusC (numC 5) (multC (numC -1) (numC 6)))
> (parse '(~ 4))
- ArithS
(uminusS (numS 4))
```