

Linguagens de Programação

Semântica Formal

Andrei Rimsa Álvares
andrei@cefetmg.br



Sumário

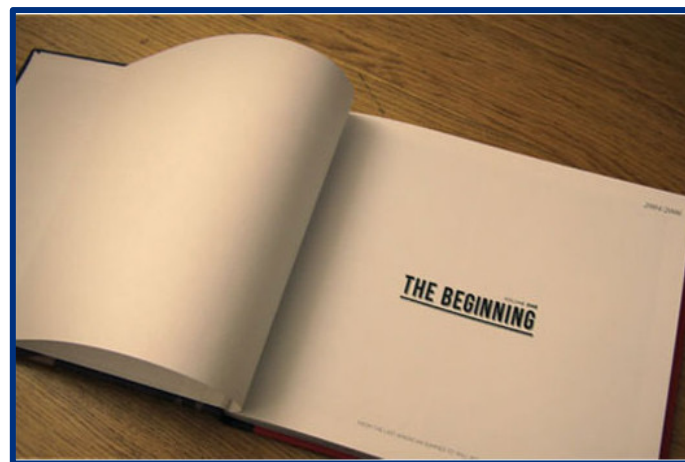
- Introdução
- Linguagem One
- Linguagem Two
- Linguagem Three



CEFET-MG

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

INTRODUÇÃO



Linguagens de Programação



Introdução

- Não existe uma única forma aceitável para definição da **semântica** de uma linguagem de programação
- Existem muitos critérios que devem ser considerados para criação de uma metodologia e definição da notação para a semântica
 - Programadores precisam entender o que os comandos significam
 - Desenvolvedores de compiladores precisam saber exatamente o que as construções da linguagem fazem
 - Provas de corretude deveriam ser possíveis
 - Geradores de compiladores deveriam ser suportados
 - Projetistas poderiam detectar ambiguidades e inconsistências

Como definir rigorosamente a semântica de uma linguagem de programação?



Semântica Formal

- Existem vários formalismos para definir formalmente a semântica de uma linguagem
 - Semântica **operacional**: define o programa como sua execução em uma máquina abstrata interpretada
 - Semântica **axiomática**: define o programa através de asserções sobre como os estados são alterados em cada passo
 - Semântica **denotacional**: tenta capturar o que a computação (de forma matemática) faz em cada passo

Vamos ver uma variação da semântica operacional mais abstrata conhecida como semântica natural



CEFET-MG

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

LINGUAGEM ONE



Linguagens de Programação



Linguagem One

- A linguagem One define expressões aritméticas usando constantes, operadores binários infixos de adição e multiplicação, com as corretas precedências e associatividades, e agrupamentos com parênteses
 - Sua estrutura léxica é dada pelos tokens: +, -, (,) e constantes inteiras contendo um dígito decimal ou mais
 - Sua estrutura sintática é dada pela seguinte gramática

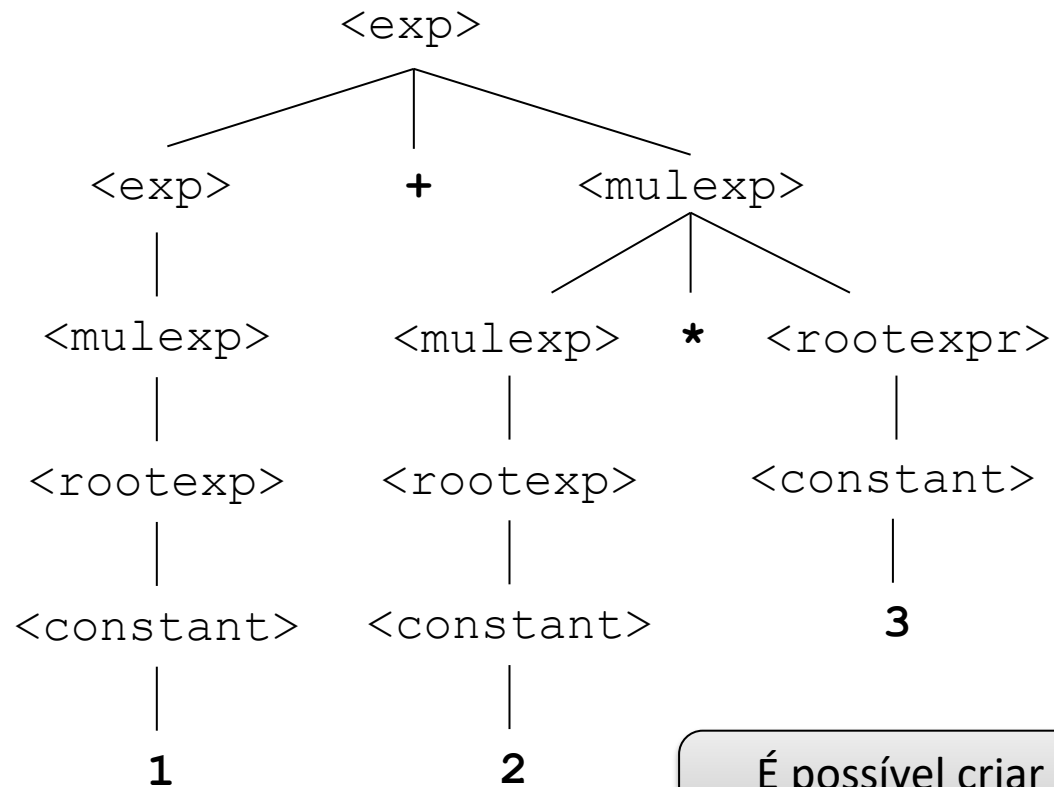
```
<exp>      ::= <exp> + <mulexp>
              | <mulexp>
<mulexp>    ::= <mulexp> * <rootexp>
              | <rootexp>
<rootexp>  ::= ( <exp> )
              | <constant>
```

Esta gramática é ambígua? Qual a árvore de derivação para a expressão $1+2*3$?



Árvore de Derivação

- A árvore de derivação para a expressão $1+2*3$ é dada a seguir

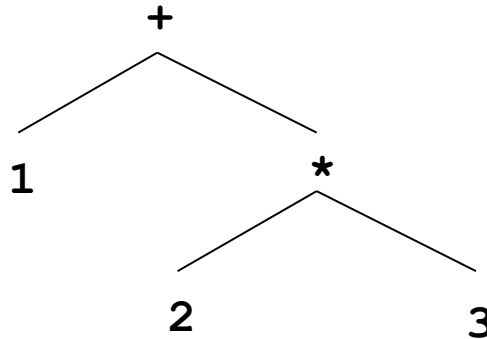


É possível criar uma versão mais compactada desta árvore?



Árvore de Sintaxe Abstrata

- A árvore de sintaxe abstrata (AST: *Abstract Syntax Tree*) é uma forma simplificada de uma árvore de derivação, mas somente com terminais



- Essa árvore poderia ser expressa linearmente da seguinte maneira

plus(const(1), times(const(2), const(3)))

Como definir a semântica para esta AST?



Árvore de Sintaxe Abstrata

- Uma forma de definir a semântica de uma linguagem é mostrando um interpretador para ela, onde a AST é a entrada para esse interpretador
- Inclusive, pode-se usar uma gramática para definir a estrutura da AST

```
<exp> ::= plus ( <exp> , <exp> )  
        | times ( <exp> , <exp> )  
        | const ( <constant> )
```

Note que a gramática para uma AST pode ser ambígua: a ordem de avaliação já foi fixado pelo parser da gramática da linguagem original

É possível fazer um interpretador para esta linguagem da AST em SML?



Interpretador em SML

- A seguir é mostrado um interpretador em SML

```
datatype exp = const of int |  
              plus of exp * exp |  
              times of exp * exp;  
  
fun vall (const x) = x  
|     vall (plus (x,y)) = vall x + vall y  
|     vall (times (x,y)) = vall x * vall y;
```

- Exemplos

```
- vall (const(1)) ;  
val it = 1 : int  
- vall (plus(const(1),const(2))) ;  
val it = 3 : int  
- vall (plus(const(1),times(const(2),const(3)))) ;  
val it = 7 : int
```

E em Prolog, é possível criar um interpretador para esta mesma linguagem?



Interpretador em Prolog

- A seguir é mostrado um interpretador em Prolog

```
val1(const(X), X).
```

```
val1(plus(X, Y), Value) :- val1(X, XValue), val1(Y, YValue),  
                           Value is XValue + YValue.
```

```
val1(times(X, Y), Value) :- val1(X, XValue), val1(Y, YValue),  
                           Value is XValue * YValue.
```

- Exemplos

```
?- val1(const(1), R).
```

```
R = 1.
```

```
?- val1(plus(const(1), const(2)), R).
```

```
R = 3.
```

```
?- val1(plus(const(1), times(const(2), const(3))), R).
```

```
R = 7.
```

Qual o problema de se criar um interpretador usando uma linguagem de programação?



Problema

- Qual deveria ser o valor de uma constante?
 - Em SML

```
- val1(const(10000000000000000000));  
stdIn:1.13-8.4 Error: literal '10000000000000000000'  
is too large for type int
```

A linguagem One
deveria fazer isto?

- Em Prolog

```
?- val1(const(2147483648),R) .
```

```
R = 2147483648 .
```

```
?- val1(plus(const(2147483648),const(1)),R) .
```

```
R = 2.14748e+009 .
```

Em versões
mais antigas

```
?- val1(const(2147483648),R) .
```

```
R = 2147483648 .
```

```
?- val1(plus(const(2147483648),const(1)),R) .
```

```
R = 2147483649 .
```

Em versões
mais novas



Definindo a Semântica via Interpretação

- A implementação `val1` não é uma definição satisfatória da semântica da linguagem One
 - Programas na linguagem One se comportam como o interpretador diz que eles se comportam: executam sobre o sistema de linguagens de SML ou Prolog por exemplo

Como resolver este problema?



Semântica Natural

- Existe uma notação formal que consegue capturar as mesmas regras de provas básicas usadas por `val1`, mas sem depender da implementação de um sistema de linguagens como SML ou Prolog
- A semântica deve definir a relação entre a AST e o seu resultado quando avaliado
 - $E \rightarrow v$ significa que a avaliação da AST E produz o valor final v , por exemplo $times(const(2), const(3)) \rightarrow 6$

Como definir regras usando semântica natural?



Regras em Semântica Natural

- A multiplicação (*times*) pode ser expressa como uma regra em semântica natural da seguinte forma, onde condições ficam acima da linha e conclusões abaixo

$$\frac{E_1 \rightarrow v_1 \quad E_2 \rightarrow v_2}{times(E_1, E_2) \rightarrow v_1 \times v_2}$$

- Pode haver mais de uma regra para um tipo de nó da AST, por exemplo para uma construção similar ao *if-then-else* de ML

$$\frac{E_1 \rightarrow true \quad E_2 \rightarrow v_2}{if(E_1, E_2, E_3) \rightarrow v_2}$$
$$\frac{E_1 \rightarrow false \quad E_3 \rightarrow v_3}{if(E_1, E_2, E_3) \rightarrow v_3}$$

Qual o conjunto de regras que definem a semântica natural da linguagem One?



Semântica Natural da Linguagem One

- A regra de multiplicação (*times*) pode ser expressa como uma regra em semântica natural da seguinte forma, onde condições ficam acima da linha e conclusões abaixo

```
val1(const(X), X).  
  
val1(plus(X, Y), Value) :-  
    val1(X, XValue),  
    val1(Y, YValue),  
    Value is XValue + YValue.  
  
val1(times(X, Y), Value) :-  
    val1(X, XValue),  
    val1(Y, YValue),  
    Value is XValue * YValue.
```

Interpretador em Prolog

VS

$$\frac{const(n) \rightarrow eval(n)}{}$$
$$\frac{E_1 \rightarrow v_1 \quad E_2 \rightarrow v_2}{plus(E_1, E_2) \rightarrow v_1 + v_2}$$
$$\frac{E_1 \rightarrow v_1 \quad E_2 \rightarrow v_2}{times(E_1, E_2) \rightarrow v_1 \times v_2}$$

Interpretador em Semântica Natural

Ainda seriam necessárias as definições de $+$, \times e *eval*, mas pelo menos não se usaria acidentalmente as definições de Prolog



CEFET-MG

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

LINGUAGEM TWO



Linguagens de Programação



Linguagem Two

- A linguagem One será estendida para adicionar as seguintes funcionalidades
 - Variáveis
 - Expressões *let* no estilo de ML para definição destas variáveis
- A sintaxe pode ser definida pela gramática

```
<exp>      ::= <exp> + <mulexp>
              | <mulexp>
<mulexp>   ::= <mulexp> * <rootexp>
              | <rootexp>
<rootexp> ::= let val <variable> = <exp> in <exp> end
              | ( <exp> ) | <variable> | <constant>
```

Como seria a AST nesta linguagem para a expressão: `let val y = 3 in y*y end`?



Árvore de Sintaxe Abstrata

- Será preciso adicionar mais dois tipos de nós na AST
 - $var(x)$ para referências a variável x
 - $let(x, exp_1, exp_2)$ para uma expressão let que avalia exp_2 em um ambiente onde a variável x foi ligada ao valor exp_1
- Assim, para o programa `let val y = 3 in y*y end` têm-se a seguinte AST

$let(y, const(3), times(var(y), var(y)))$

Onde os valores das variáveis são armazenados? Como recuperá-los?



Contexto/Ambiente

- Um valor é ligado a uma variável pelo predicado em Prolog `bind(Variable, Value)`
- Um contexto em irá armazenar o ambiente das variáveis; em Prolog é uma lista de zero ou mais termos de ligação (`bind`)
 - O contexto onde `x` é ligado ao valor `3` poderia ser `[bind(y, 3)]`
 - Já o contexto onde ambos `x` e `y` são ligados ao valor `3` poderia ser `[bind(x, 3), bind(y, 3)]`
- Pode-se criar um predicado em Prolog para buscar (`lookup`) uma variável em um contexto da seguinte maneira

```
lookup(Variable, [bind(Variable, Value)|_], Value) :- !.  
lookup(Variable, [_|Rest], Value) :-  
    lookup(Variable, Rest, Value).
```

Como criar um interpretador para a linguagem Two em Prolog?



Interpretador em Prolog

- A seguir é mostrado um interpretador em Prolog

```
val2(const(X), Context, X).
```

```
val2(plus(X, Y), Context, Value) :- val2(X, Context, XValue),  
    val2(Y, Context, YValue),  
    Value is XValue + YValue.
```

```
val2(times(X, Y), Context, Value) :- val2(X, Context, XValue),  
    val2(Y, Context, YValue),  
    Value is XValue * YValue.
```

```
val2(var(X), Context, Value) :- lookup(X, Context, Value).
```

```
val2(let(X, Exp1, Exp2), Context, Value2) :-  
    val2(Exp1, Context, Value1),  
    val2(Exp2, [bind(X, Value1)|Context], Value2).
```



Interpretador em Prolog

- Exemplo

```
let val y = 3 in y*y end
```

```
let (y, const (3), times (var (y), var (y)))
```

```
?- val2 (let (y, const (3), times (var (y), var (y))), nil, R).  
R = 9.
```



Interpretador em Prolog

- Exemplo

```
let val y = 3 in
  let val x = y*y in
    x*x
  end
end
```

let (y, const (3), let (x, times (var (y), var (y)), times (var (x), var (x))))

```
?- val2(let(y, const(3), let(x, times(var(y), var(y)),
|   times(var(x), var(x))), nil, R).
R = 81.
```




Interpretador em Prolog

- Exemplo

```
let val y = 1 in  
  let val y = 2 in  
    y  
  end  
end
```

let (y, const (1), let (y, const (2), var (y)))

```
?- val2(let(y, const(1), let(y, const(2), var(y))), nil, R).  
R = 2.
```

Como criar um interpretador para a linguagem Two em semântica natural?



Interpretador em Semântica Natural

- A seguir é mostrado um interpretador em semântica natural
 - $\langle E, C \rangle \rightarrow v$ significa que a avaliação da AST E no contexto C produz o valor v

$$\langle \text{const}(n), C \rangle \rightarrow \text{eval}(n)$$

$$\frac{\langle E_1, C \rangle \rightarrow v_1 \quad \langle E_2, C \rangle \rightarrow v_2}{\langle \text{plus}(E_1, E_2), C \rangle \rightarrow v_1 + v_2}$$

$$\frac{\langle E_1, C \rangle \rightarrow v_1 \quad \langle E_2, C \rangle \rightarrow v_2}{\langle \text{times}(E_1, E_2), C \rangle \rightarrow v_1 \times v_2}$$

$$\langle \text{var}(v), C \rangle \rightarrow \text{lookup}(v, C)$$

$$\frac{\langle E_1, C \rangle \rightarrow v_1 \quad \langle E_2, \text{bind}(x, v_1) :: C \rangle \rightarrow v_2}{\langle \text{let}(x, E_1, E_2), C \rangle \rightarrow v_2}$$

Para que esta semântica ficasse completa, ainda seria preciso definir $+$, \times , eval e as novas funções lookup , bind , $::$ (cons), e o contexto inicial nil



Sobre Erros

- Na linguagem One, todos os programas sintaticamente corretos executavam sem erros
- Porém, isto não é verdade para a linguagem Two, por exemplo

```
let val a = 1 in b end
```

Qual a semântica desta construção?

– Em Prolog

```
?- val2(let(a, const(1), var(b)), nil, X).  
false.
```

– Em Semântica Natural

$\langle \text{let}(a, \text{const}(1), \text{var}(b)), \text{nil} \rangle \rightarrow v$

Não existe v que satisfaça
 $\langle \text{let}(a, \text{const}(1), \text{var}(b)), \text{nil} \rangle$



Semântica Estática e Dinâmica

- Normalmente, sistemas de linguagens fazem verificação de erros depois do *parsing*, mas antes de executar
 - Para escopo estático: referências precisam estar no escopo de alguma definição desta variável
 - Para tipagem estática: uma forma consistente de atribuir tipos a todas as partes do programa
- A semântica natural que foi especificada até agora se trata apenas de semântica dinâmica (a semântica estática já pegou alguns dos erros)
 - Porém, problemas ainda podem ocorrer em tempo de execução

Não é sintaxe nem comportamento da execução

Como lidar com estes erros?

$\langle \text{divide}(\text{const}(6), \text{const}(3)), C \rangle \rightarrow \langle \text{normal}, 2 \rangle$
 $\langle \text{divide}(\text{const}(6), \text{const}(0)), C \rangle \rightarrow \langle \text{abrupt}, \text{dividebyzero} \rangle$



CEFET-MG

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

LINGUAGEM THREE



Linguagens de Programação



Linguagem Three

- A linguagem Two será estendida para adicionar as seguintes funcionalidades no estilo de ML
 - Valores funções
 - Aplicação de funções
- A sintaxe pode ser definida pela gramática

```
<exp> ::= fn <variable> => <exp> | <addexp>
<addexp> ::= <addexp> + <mulexp> | <mulexp>
<mulexp> ::= <mulexp> * <funexp> | <funexp>
<funexp> ::= <funexp> <rootexp> | <rootexp>
<rootexp> ::= let val <variable> = <exp> in <exp> end
           | ( <exp> ) | <variable> | <constant>
```

Como seria a AST nesta linguagem para a expressão: `(fn x => x * x) 3`?



Árvore de Sintaxe Abstrata

- Será preciso adicionar mais dois tipos de nós na AST
 - $apply(Function, Actual)$ para aplicar a função ($Function$) ao parâmetro atual ($Actual$)
 - $fn(Formal, Body)$ para uma expressão de função (fn) com o parâmetro formal ($Formal$) e o corpo ($Body$)
- Assim, para o programa $(fn\ x\ =>\ x*x)\ 3$ têm-se a seguinte AST

$apply(fn(x, times(var(x), var(x))), const(3))$

Como representar funções?



Funções

- Uma função é representada em Prolog pelo predicado `fval (Formal, Body)`, onde
 - `Formal` é a variável do parâmetro formal
 - `Body` é o o corpo não avaliado da função
- Portanto, a avaliação do nó da AST `fn (Formal, Body)` resulta em `fval (Formal, Body)`

Por que criar uma nova representação (`fval`) ao invés de usar o próprio nó da AST para funções (`fn`)?

Como criar um interpretador para a linguagem Three em Prolog?



Interpretador em Prolog

- A seguir é mostrado um interpretador em Prolog

```
% Os mesmos predicados da linguagem Two
val3(const(X), Context, X).
val3(plus(X, Y), Context, Value) :- ...
val3(times(X, Y), Context, Value) :- ...
val3(var(X), Context, Value) :- ...
val3(let(X, Exp1, Exp2), Context, Value2) :- ...

% Mais os novos predicados para a linguagem Three
val3(fn(Formal, Body), Context, fval(Formal, Body)).

val3(apply(Function, Actual), Context, Value) :-
    val3(Function, Context, fval(Formal, Body)),
    val3(Actual, Context, ParamValue),
    val3(Body, [bind(Formal, ParamValue)|Context], Value).
```



Interpretador em Prolog

- Exemplo

```
(fn x => x * x) 3
```

```
apply(fn(x, times(var(x), var(x))), const(3))
```

```
?- val3(apply(fn(x, times(var(x), var(x))), const(3)), nil, R) .  
R = 9.
```



Interpretador em Prolog

- Exemplo

```
let val x = 1 in
  let val f = fn n => n + x in
    let val x = 2 in
      f 0
    end
  end
end
```

*let(x, const(1), let(f, fn(n, plus(var(n), var(x))),
let(x, const(2), apply(var(f), const(0))))))*

```
?- val3(let(x, const(1), let(f, fn(n, plus(var(n), var(x))),  
| let(x, const(2), apply(var(f), const(0))))), nil, R).  
R = 2.
```

Qual escopo foi implementado aqui? Este escopo está correto?



Funções com Escopo Estático

- No escopo estático, deve-se adicionar um contexto à função em Prolog pelo predicado `fval (Formal, Context, Body)`, onde
 - `Formal` é a variável do parâmetro formal
 - `Context` é o contexto a ser usado na chamada
 - `Body` é o o corpo não avaliado da função
- Portanto, a avaliação do nó da AST `fn (Formal, Body)` no contexto `Context` resulta em `fval (Formal, Context, Body)`

Neste caso, o contexto (`Context`) funciona como um link de encadeamento

Como criar refazer o interpretador em Prolog para usar escopo estático?



Interpretador em Prolog com Escopo Estático

- A seguir é mostrado um interpretador em Prolog

```
val3(fn(Formal, Body), Context, fval(Formal, Body)).
```



```
val3(fn(Formal, Body), Context, fval(Formal, Context, Body)).
```

```
val3(apply(Function, Actual), Context, Value) :-  
  val3(Function, Context, fval(Formal, Body)),  
  val3(Actual, Context, ParamValue),  
  val3(Body, [bind(Formal, ParamValue)|Context], Value).
```



```
val3(apply(Function, Actual), Context, Value) :-  
  val3(Function, Context, fval(Formal, Nesting, Body)),  
  val3(Actual, Context, ParamValue),  
  val3(Body, [bind(Formal, ParamValue)|Nesting], Value).
```



Interpretador em Prolog

- Exemplo

```
let val x = 1 in
  let val f = fn n => n + x in
    let val x = 2 in
      f 0
    end
  end
end
```

*let(x, const(1), let(f, fn(n, plus(var(n), var(x))),
let(x, const(2), apply(var(f), const(0))))))*

```
?- val3(let(x, const(1), let(f, fn(n, plus(var(n), var(x))),  
| let(x, const(2), apply(var(f), const(0))))), nil, R).  
R = 1.
```

Agora com
escopo estático



Interpretador em Prolog

- Exemplo

```
let
  val f = fn x =>
    let
      val g = fn y => y+x in g
    end
in
  f 1 2
end
```

*let(f, fn(x, let(g, fn(y, plus(var(y), var(x))), var(g))),
apply(apply(var(f), const(1)), const(2))), nil, X)*

```
?- val3(let(f, fn(x, let(g, fn(y, plus(var(y), var(x))), var(g))),  
|   apply(apply(var(f), const(1)), const(2))), nil, X).  
X = 3.
```

Inclusive funciona com funções
de ordem superior e *currying*

Como criar um interpretador para a
linguagem Three em semântica natural?



Interpretador em Semântica Natural com Escopo Dinâmico

- A seguir é mostrado um interpretador em semântica natural

$$\langle \text{const}(n), C \rangle \rightarrow \text{eval}(n)$$

$$\frac{\langle E_1, C \rangle \rightarrow v_1 \quad \langle E_2, C \rangle \rightarrow v_2}{\langle \text{plus}(E_1, E_2), C \rangle \rightarrow v_1 + v_2}$$

$$\frac{\langle E_1, C \rangle \rightarrow v_1 \quad \langle E_2, C \rangle \rightarrow v_2}{\langle \text{times}(E_1, E_2), C \rangle \rightarrow v_1 \times v_2}$$

$$\langle \text{var}(v), C \rangle \rightarrow \text{lookup}(v, C)$$

$$\frac{\langle E_1, C \rangle \rightarrow v_1 \quad \langle E_2, \text{bind}(x, v_1) :: C \rangle \rightarrow v_2}{\langle \text{let}(x, E_1, E_2), C \rangle \rightarrow v_2}$$

$$\langle \text{fn}(x, E), C \rangle \rightarrow (x, E)$$

$$\frac{\langle E_1, C \rangle \rightarrow (x, E_3) \quad \langle E_2, C \rangle \rightarrow v_1 \quad \langle E_3, \text{bind}(x, v_1) :: C \rangle \rightarrow v_2}{\langle \text{apply}(E_1, E_2), C \rangle \rightarrow v_2}$$

E como seria com escopo estático?



Interpretador em Semântica Natural com Escopo Estático

- A seguir é mostrado um interpretador em semântica natural

$$\langle fn(x, E), C \rangle \rightarrow (x, E)$$



$$\langle fn(x, E), C \rangle \rightarrow (x, C, E)$$

$$\frac{\langle E_1, C \rangle \rightarrow (x, E_3) \quad \langle E_2, C \rangle \rightarrow v_1 \quad \langle E_3, bind(x, v_1) :: C \rangle \rightarrow v_2}{\langle apply(E_1, E_2), C \rangle \rightarrow v_2}$$



$$\frac{\langle E_1, C \rangle \rightarrow (x, C', E_3) \quad \langle E_2, C \rangle \rightarrow v_1 \quad \langle E_3, bind(x, v_1) :: C' \rangle \rightarrow v_2}{\langle apply(E_1, E_2), C \rangle \rightarrow v_2}$$



Sobre Erros

- A linguagem Three agora possui dois tipos (constantes e funções), portanto é possível ter erros de tipos

1 1

Qual a semântica desta aplicação?

- Em Prolog

```
?- val3(apply(const(1), const(1)), nil, X).  
false.
```

- Em Semântica Natural

$\langle \text{apply}(\text{const}(1), \text{const}(1)), \text{nil} \rangle \rightarrow v$

Não existe v que satisfaça
 $\langle \text{apply}(\text{const}(1), \text{const}(1)), \text{nil} \rangle$



Mais Erros

- Na versão da linguagem com escopo dinâmico, existem programas que executam para sempre, como por exemplo

```
let val f = fn x => f x in f 1 end
```

Qual a semântica deste programa?

- Em Prolog: interpretador executa para sempre

```
?- val3(let(f,fn(x,apply(var(f),const(1))),  
|      apply(var(f),const(1))),nil,R).  
*** não termina ***
```

- Em Semântica Natural: não executa para sempre, já que nem chega a interpretar; ela simplesmente não gera resultado

$\langle \text{let}(f, \text{fn}(x, \text{times}(\text{var}(x), \text{const}(1))), \text{apply}(\text{var}(f), \text{const}(1))), \text{nil} \rangle \rightarrow v$

Não existe v que satisfaça

$\langle \text{let}(f, \text{fn}(x, \text{times}(\text{var}(x), \text{const}(1))), \text{apply}(\text{var}(f), \text{const}(1))), \text{nil} \rangle$



CEFET-MG

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

ISSO É TUDO, PESSOAL!



Linguagens de Programação