

Especificação da Linguagem nonNeumann

1. Introdução

Indo contra o padrão de arquitetura de computadores vigente, a linguagem nonNeumann é uma linguagem funcional que contém características – muito simplificadas e aleatórias - de ML, Scheme, LISP e Haskell.

Ela apresenta as seguintes características principais:

Tipos primitivos: int, real, bool;

Tipos compostos: não possui;

Estruturas de controle: if-else, recursão;

Escopo das constantes: escopo do bloco função;

Funções: há suporte a funções padrão, lambda e de alta ordem;

Comentários: bloco e linha.

nonNeumann, obviamente, é uma linguagem experimental, então esta especificação é passível de adaptações. Modificações na especificação serão notificadas via SIGAA e Google Classroom.

2. Léxico

- Comentários de bloco são delimitados por “{-” e “-}”. O comentário de linha é feito com “;” no início da linha.

- Identificadores podem conter um ou mais letras ou underlines, em qualquer ordem.

- Números inteiros podem ser escritos em decimal ou em hexadecimal (começando com 0x)

- Números reais podem ser escritos em decimal ou notação científica (com E apenas). Os valores em decimal devem ser separados por ponto:

Exemplo: $5E-3 = 5 * 10^{-3} = 0.005$

Vide Seção 3 para mais informações sobre as classes de tokens a serem implementadas nesta etapa.

3. Sintático

A gramática abaixo foi escrita em uma versão de E-BNF seguindo as seguintes convenções:

- 1 - variáveis da gramática são escritas em letras minúsculas sem aspas.
- 2 - lexemas que correspondem diretamente a tokens são escritos entre aspas simples
- 3 - símbolos escritos em letras maiúsculas representam o lexema de um token do tipo especificado.
- 4 - o símbolo | indica produções diferentes de uma mesma variável.
- 5 - o operador [] indica uma estrutura sintática opcional (similar à ?).
- 6 - o operador { } indica uma estrutura sintática que é repetida zero ou mais vezes (fecho).

```
programa : { dec_funcao }

dec_funcao : ['entry'] '(' 'function' tipo ID '(' parametros ')' ':'
            '(' exp ')' ')'

tipo : 'int' | 'bool' | 'real'

parametros : ε | parametro { ',' parametro }
parametro : tipo ID | assinatura
assinatura : tipo ID '(' parametros_assinatura ')'

//os parâmetros na assinatura de uma função passada como parametro não têm
//identificador
parametros_assinatura : ε | parametro_assinatura { ',' parametro_assinatura }
parametro_assinatura : tipo | assinatura

exp : NUMERAL_INT | NUMERAL_REAL | 'true' | 'false'
    | bloco_exp
    | chamada
    | lambda
    | '-' exp
    | 'if' '(' exp ')' 'then' exp 'else' exp
    | exp '+' exp
    | exp '-' exp
    | exp '*' exp
    | exp '/' exp
    | exp '%' exp
    | exp '=' exp
    | exp '<' exp
    | exp '>' exp
    | '!' exp
    | exp 'and' exp
    | exp 'or' exp
bloco_exp : '(' { dec_cons } exp ')'
dec_cons : '(' 'const' tipo ID '(' exp ')' ')'

chamada : ID '(' lista_exp ')'
lambda : 'lambda' '(' lista_ids ')' : '(' exp ')' [ '[' lista_exp ']' ]
lista_ids : ε | ID { ',' ID }
lista_exp : ε | exp { ',' exp }
```

4. Semântico:

- A execução de um programa consiste na execução de uma função com modificador “entry”. A inexistência de um ponto de entrada não impede a compilação, mas gera um *warning*.

- A precedência dos operadores é a matemática.

- Em operações entre os tipos int e real, os valores int devem ser convertidos para real.

- É possível criar funções lambda com tipo implícito. Exemplo:

lambda (y) : (y*y) [7] => expressão que vale 49

Obs. O tempo de vida do identificador y é o mesmo da função lambda.

- Funções de alta ordem em nonNeumann: uma função pode receber uma ou mais funções como parâmetros. Exemplo:

```
( function bool max (int x, int y) : (if (x>y) then x else y))  
( function bool apply_operator (int x, int y, bool op(int,int) ) : (op(x, y) ) ; função de alta ordem  
entry ( function bool principal () : (apply_operator(7, 5, max) )
```

```
( function int max (int x, int y) : (if (x>y) then x else y))  
( function int apply_operator (int x, int y, int op(int,int) ) : (op(x, y) ) ; função de alta ordem  
entry ( function int principal () : (apply_operator(7, 5, max) )
```

O que deve ser verificado na análise semântica:

- Se as entidades criadas pelo usuário são inseridas na tabela de símbolos - com os atributos necessários - quando são declaradas;
- Se uma entidade foi declarada e está em um escopo válido no momento em que ela é utilizada;
- Se entidades foram definidas (inicializadas) quando isso se fizer necessário;
- Checar a compatibilidade dos tipos de dados envolvidos nas **declarações** e **expressões**

5. Desenvolvimento do Trabalho

Trabalhos devem ser desenvolvidos em trio, dupla ou individualmente. Será aberto um fórum no Classroom para a discussão sobre as etapas. Em caso de dúvida, verifique inicialmente no fórum se ela já foi resolvida. Se ela persiste, consulte a professora.

5.1. Ferramentas

- Submissão das etapas do projeto: **via SIGAA**. Será criada uma ou mais tarefas para cada etapa.
- Implementação com SableCC e Java.

5.2. Avaliação

- A avaliação será feita com base nas etapas entregues e em arguições feitas com os grupos.
- A aula que define o prazo de entrega de cada etapa será especificada no plano de curso da disciplina.

5.3. Etapas

Análise Léxica (valor: 2.5)

- Tarefa 1: três códigos em nonNeumann que, unidos, usem todas as alternativas gramaticais (ou seja, todos os recursos) da linguagem.
- Tarefa 2: Analisador léxico em SableCC, com impressão dos lexemas e tokens reconhecidos ou impressão dos erros

Análise Sintática (valor: 2.5):

- Tarefa 1: analisador sintático em SableCC, com impressão da árvore sintática em caso de sucesso ou impressão dos erros

Sintaxe Abstrata (valor: 2.5):

- Tarefa 1: analisador sintático abstrato em SableCC, com impressão da árvore sintática

Análise Semântica (valor: 2.5):

- Tarefa 1: criação e impressão da tabela de símbolos
- Tarefa 2: validação de escopo e de existência de identificadores. Verificação de tipos

Geração de código (extra: 2.0):

- Código em linguagem alvo (C)

A critério da docente o valor das etapas pode ser modificado, desde que este novo cálculo produza uma nota não menor que a produzida pelo cálculo original. Entregas após o prazo sofrem penalidade de meio ponto por dia de atraso.

Bom trabalho!