Trabalho 3 - Analisador semântico

Nathan Reuter Godinho
Ad Nunes Ribeiro
Eduardo Dias

Instruções para Rodar o Código:

1 - Entre na pasta **t1comp-master/**
2 **-** Rode o comando no terminal para compilar: $ **make**
3 - Rode o comando para executar o programa: $ **make run**

Inicialmente para realizar as tarefas Sem2 e GCI2 a gramática foi reduzida para se adequar a gramática STAT descrita removendo todas as produções que não fossem produtivas a partir de **STATEMENT** além de adicionar uma nova produção "**PROGRAM**->**STATEMENT**" onde **PROGRAM** é o símbolo inicial da gramática.

Quanto os 3 programas solicitados, eles podem ser acessados na no caminho "t1comp/src/testCodes/", onde se encontram os exemplos de todas as etapas, e são identificados com a inicial exemploT3:

1. exemploT3.xpp
2. exemploT3_1.xpp
3. exemploT3_2.xpp

Tarefa ASem2

A solução para os itens da tarefa foi feita durante a criação da árvore semântica, analisando os tokens passados pelo analisador léxico. Além da verificação do token atual, o analisador também analisa os tokens adjacentes para verificar o tipo da variável. Vale a pena lembrar que o analisador foi implementado considerando no perfeito funcionamento do analisador léxico. Abaixo segue a explicação de cada item da tarefa.

Escopos

Os escopos foram feitos através de hashs, mais especificamente em dois hashs que guardam respectivamente:

1. Tabela com as variáveis
2. Tabela com os tipos

Ambas as tabelas possuem como chave o identificador da variável, facilitando a recuperação da informação. Além disso cada escopo possui uma variável booleana que informa se o escopo está dentro de um comando de repetição.

O controle de escopo é feito através de uma tabela de escopos e uma pilha, na qual é feito o empilhamento e desempilhamento de acordo com a chegada de determinados tokens. A definição considera como escopo tudo aquilo entre chaves("{}"), tais tokens serão usados para tirar ou colocar na pilha.

- Caso apareça ("{"):
    - Cria novo escopo, setando a variável booleana de acordo com a presença ou não do token "FOR" anteriormente.
    - Adiciona escopo a tabela de escopos.
    - Empilha escopo antigo.
- Caso apareça ("}"):
    - Remove o escopo da pilha (tal escopo se torna o escopo atual).

Declaração de Variáveis

Ao receber um novo token do analisador léxico, o analisador verifica se o tipo corresponde à um identificador "ident", se tal suposição for confirmada ele busca o nome do token e passa a verificar tokens adjacentes (token anterior e tokens futuros):

- O analisador desconsidera o token caso o token seguinte a este for outro identificador e não faz nada.
- O analisador não considera como inicialização de variável caso o token anterior for algum token de sinal aritmético (como "=", "+", "-", "*", "/"), ou sinal de pontuação (como "()","[]") ou algum comando.

- Caso a lista de variáveis do escopo atual já contém a variável, o analisador retorna erro de declaração de variáveis. Caso contrário ele passa a verificar o tipo da variável (descrito no próximo item)
- Durante a declaração da variável, já é realizada a verificação de tipo. Para isto o analisador verifica o token anterior ao identificador. Se o token for um identificador, o nome deste passa a se tornar o tipo da variável.
- Após a verificação de tipo, o analisador adiciona o nome do token à lista de variáveis do escopo juntamente com o tipo. Além de atualizar a tabela de ecopos.

Nenhuma variável apresentou algum problema, assim como é impossível adicionar uma variável repetida.

Verificação de Tipo

Durante a declaração da variável, já é realizada a verificação de tipo. Para isto o analisador verifica o token anterior ao identificador. Se o token for um identificador ou nome do tipo ("int","string"), o nome deste passa a se tornar o tipo da variável.

Caso o token anterior for uma vírgula significa que é uma sequência de variáveis sendo declaradas em sequência, dessa forma o analisador atribui o último tipo que foi declarado.

Break

Como foi descrito anteriormente, toda estrutura dos escopos já estão montadas, facilitando esta verificação. Basicamente o analisador ao receber um token "break" verifica a variável booleana do escopo atual:

- Caso true, retorna o erro.

Para todos os casos, não foi encontrado forma do comando estar fora do loop, sem que o analisador acuse o erro.

Tarefa GCI2

Sobre a gramática STAT existente foram adicionados atributos, respeitando os limites de uma sdd L-atribuída. Devido às mudanças na

gramática algumas partes de declaração de variáveis foi alterada para se adequar a gramática.

Deve ser dado destaque as labels, que são globais, podem ser usadas como **STATEMENT.return** e **STATEMENT.break** para definir labels q não precisam ser necessariamente passadas entre as produções.

O texto em azul abaixo de cada produção representa os atributos relacionados a geração de código intermediário. Para melhor entendimento das atribuições considere "||" como sendo a concatenação, "gen" como sendo a geração de uma linha de código de três endereços.

O texto em vermelho abaixo de cada produção representa os atributos herdados do nodo pai, enquanto os em rosa representam atributos herdados do irmão da esquerda.

Quanto à execução, a geração de labels e temporários acontece em tempo de execução contudo a avaliação dos atributos segue a avaliação dos nodos.

A Gramatica completa pode ser observada abaixo.

Gramatica STAT

PROGRAM -> STATEMENT
STATEMENT.next = newLabel();
STATEMENT.return = STATEMENT.next()
PROGRAM.code = STATEMENT.code || label(STATEMENT.next)|| gen(exit)
STATEMENT -> **PRINTSTAT** ;
PRINTSTAT.next = STATEMENT.next
STATEMENT.code = PRINTSTAT.code || label(PRINTSTAT.next)
STATEMENT -> **READSTAT** ;
READSTAT.next = STATEMENT.next
STATEMENT.code = READSTAT.code || label(READSTAT.next)
STATEMENT -> **RETURNSTAT** ;
STATEMENT.code = RETURNSTAT.code
STATEMENT -> **IFSTAT**
IFSTAT.next = STATEMENT.next
STATEMENT.code = IFSTAT.code || label(IFSTAT.next)

STATEMENT -> FORSTAT
FORSTAT.next = FORSTAT.next
STATEMENT.code = FORSTAT.code || label(FORSTAT.next)
STATEMENT -> { STATLIST }
STATEMENT.code = STATLIST.code
STATLIST.next = STATMENT.next
STATEMENT -> break ;
STATEMENT.code = gen('goto' STATEMENT.break)
STATEMENT -> ;
STATEMENT.code = ''
STATEMENT -> int ident VARDECL1 ;
VARDECL1.her = new leaf(id,"int")
addType(ident,VARDECL1.sin)
STATEMENT.width = '4'
STATEMENT.aloc = newTemp()
STATEMENT.code = VARDECL1.code
|| gen(STATEMENT.aloc '=' STATEMENT.width '*'
VARDECL1.width)
|| gen(aloc tabSimbolo(ident) STATEMENT.aloc)
STATEMENT -> string ident VARDECL1 ;
VARDECL1.her = new leaf(id,"string")
addType(ident,VARDECL1.sin)
STATEMENT.width = '4'
STATEMENT.aloc = newTemp()
STATEMENT.code = VARDECL1.code
|| gen(STATEMENT.aloc '=' STATEMENT.width '*'
VARDECL1.width)
|| gen(aloc tabSimbolo(ident) STATEMENT.aloc)
STATEMENT -> ident STATEMENT1
STATEMENT1.her = new leaf(id,tabSimbolo(ident))
STATEMENT.code = STATEMENT1.code
STATEMENT1 -> ident VARDECL1 ;
VARDECL1.her = STATEMENT1.her

addType(ident,VARDECL1.sin)
STATEMENT1.width = ‘4’
STATEMENT1.aloc = newTemp()
STATEMENT1.code = VARDECL1.code
|| gen(STATEMENT1.aloc ‘=’ STATEMENT1.width ‘*’
VARDECL1.width)
|| gen(aloc tabSimbolo(ident) STATEMENT1.aloc)
STATEMENT1 -> = NUMEXPRESSION ;
STATEMENT1.addr = STATEMENT1.her
STATEMENT1.code = NUMEXPRESSION.code ||
gen(STATEMENT1.addr ‘=’NUMEXPRESSION.addr)
STATLIST -> STATEMENT STATLIST1
STATEMENT.next = NewLabel()
STATLIST.code = STATEMENT.code || STATLIST1.code
STATLIST1 -> STATLIST
STATLIST1.code = STATLIST.code
STATLIST1 -> "
STATLIST1.code = ‘’
VARDECL1 -> VARDECLBRACKETS VARDECL2
VARDECLBRACKETS.her = VARDECL1.her
VARDECL2.her = VARDECL1.her
VARDECL1.sin = VARDECLBRACKETS.sin
VARDECL1.code = VARDECL2.code || VARDECLBRACKETS.code
VARDECL1.width = VARDECLBRACKETS.width
VARDECL1 -> VARDECL2
VARDECL2.her = VARDECL1.her
VARDECL1.sin = VARDECL2.sin
VARDECL1.code = VARDECL2.code
VARDECL1.width = ‘1’
VARDECL2 -> VARDECLWITHCOMA
VARDECLWITHCOMA.her = VARDECL2.her
VARDECL2.sin = VARDECL2.her
VARDECL2.code = VARDECLWITHCOMA.code

VARDECL2 -> "
VARDECL2.sin = VARDECL2.her
VARDECL2.code = ‘’
VARDECLBRACKETS -> [int-costant] VARDECLBRACKETS1
VARDECLBRACKETS1.her = VARDECLBRACKETS.her
VARDECLBRACKETS.sin = new node(new leaf(‘array’),new leaf (tabSimbolo(int-constant)),VARDECLBRACKETS1.sin)
VARDECLBRACKETS.width = newTemp()
VARDECLBRACKETS.code = VARDECLBRACKETS1.code || gen(VARDECLBRACKETS.width ‘=’VARDECLBRACKETS1.width ‘*’ tabSimbolo(int-constant))
VARDECLBRACKETS1 -> "
VARDECLBRACKETS1.sin = VARDECLBRACKETS1.her
VARDECLBRACKETS1.width = ‘1’
VARDECLBRACKETS1.code = ‘’
VARDECLBRACKETS1 -> VARDECLBRACKETS
VARDECLBRACKETS.her = VARDECLBRACKETS1.her
VARDECLBRACKETS1.sin = VARDECLBRACKETS.sin
VARDECLBRACKETS1.width = VARDECLBRACKETS.width
VARDECLBRACKETS1.code = VARDECLBRACKETS.code
VARDECLWITHCOMA -> , ident VARDECLWITHCOMA1
VARDECLWITHCOMA1.her = VARDECLWITHCOMA.her
addType(ident,VARDECLWITHCOMA1.sin)
VARDECLWITHCOMA.width = ‘4’
VARDECLWITHCOMA.aloc = newTemp()
VARDECLWITHCOMA.code = VARDECLWITHCOMA1.code
    || gen(VARDECLWITHCOMA.aloc ‘=’ VARDECLWITHCOMA.width ‘*’ VARDECLWITHCOMA1.width)
    || gen(aloc tabSimbolo(ident) VARDECLWITHCOMA.aloc)

VARDECLWITHCOMA1 -> VARDECLBRACKETS VARDECL2
VARDECLBRACKETS.her = VARDECLWITHCOMA1.her
VARDECLWITHCOMA1.sin = VARDECLBRACKETS.sin

VARDECL2.her = VARDECLWITHCOMA1.her
VARDECLWITHCOMA1.code = VARDECL2.code ||
VARDECLBRACKETS.code
VARDECLWITHCOMA1.width = VARDECLBRACKETS.width
VARDECLWITHCOMA1 -> VARDECLWITHCOMA
VARDECLWITHCOMA.her = VARDECLWITHCOMA1.her
VARDECLWITHCOMA1.sin = VARDECLWITHCOMA1.her
VARDECLWITHCOMA1.code = VARDECLWITHCOMA.code
VARDECLWITHCOMA1.width = '1'
VARDECLWITHCOMA1 -> ''
VARDECLWITHCOMA1.sin = VARDECLWITHCOMA1.her
VARDECLWITHCOMA1.code = ''
VARDECLWITHCOMA1.width = '1'
ATRIBSTAT -> LVALUE = NUMEXPRESSION
ATRIBSTAT.addr = LVALUE.addr
ATRIBSTAT.code = NUMEXPRESSION.code || LVALUE.code ||
gen(LVALUE.addr '='NUMEXPRESSION.addr)
PRINTSTAT -> print NUMEXPRESSION
PRINTSTAT.code = NUMEXPRESSION.code || gen('out'
NUMEXPRESSION.addr)
READSTAT -> read LVALUE
READSTAT.code = LVALUE.code || gen('in' LVALUE.addr)
RETURNSTAT -> return RETURNSTAT1
RETURNSTAT.code = RETURNSTAT1.code
|| gen('goto' STATEMENT.return)
RETURNSTAT1 -> NUMEXPRESSION
RETURNSTAT1.code = NUMEXPRESSION.code
RETURNSTAT1 -> ''
RETURNSTAT1.code = ''
IFSTAT -> if (NUMEXPRESSION) STATEMENT IFSTAT1
NUMEXPRESSION.true = newLabel();
NUMEXPRESSION.false = newLabel();
STATEMENT.next = IFSTAT.next

IFSTAT1.next = IFSTAT.next
IFSTAT.code = NUMEXPRESSION.code || gen(if
NUMEXPRESSION.addr '!=' '0' 'goto' NUMEXPRESSION.true)||
label(NUMEXPRESSION.false) ||IFSTAT1.code
|| label(NUMEXPRESSION.true) ||STATEMENT.code || gen('goto'
STATEMENT.next)
IFSTAT1 -> else STATEMENT end
STATEMENT.next = IFSTAT1.next
IFSTAT1.code = STATEMENT.code || gen('goto' STATEMENT.next)
IFSTAT1 -> end
IFSTAT1.code = gen('goto' IFSTAT1.next)
FORSTAT -> for (ATRIBSTAT ; NUMEXPRESSION ; ATRIBSTAT')
STATEMENT
NUMEXPRESSION.true = newLabel();
NUMEXPRESSION.false = FORSTAT.next;
FORSTAT.end = newLabel();
FORSTAT.begin = newLabel();
FORSTAT.temp = newTemp()
STATEMENT.next = FORSTAT.end
STATEMENT.break = FORSTAT.next
FORSTAT.code = ATRIBSTAT.code
|| label(FORSTAT.begin)
|| NUMEXPRESSION.code
|| gen(if NUMEXPRESSION.addr '!=' '0' 'goto' NUMEXPRESSION.true)
|| gen('goto' NUMEXPRESSION.false)
|| label(NUMEXPRESSION.true)
|| STATEMENT.code
|| label(FORSTAT.end)
|| ATRIBSTAT'.code
|| gen('goto' FORSTAT.begin)
LVALUE -> ident LVALUET2
LVALUET2.her = new leaf(tabSimbolo(ident))
LVALUE.node = LVALUET2.sin

LVALUE2.addrher = tabSimbolo(ident)
LVALUE.code = LVALUE2.code
LVALUE.addr = LVALUE2.addr
LVALUET2 ->[int-constant] LVALUET2
LVALUET2'.her = new node(new leaf('array'),new leaf(tabSimbolo(int-constant)),LVALUET2.her)
LVALUET2.sin = LVALUET2'.sin
LVALUE2.addr = newTemp()
LVALUE2.local = newTemp()
LVALUE2.code = gen(LVALUE2.local '=' tabSimbolo(int-constant) '*' '4') || gen(LVALUE2.addr '=' LVALUE2.addrher '+' LVALUE2.local) || LVALUE2'.code
LVALUE2'.addrher = LVALUE2.addr
LVALUET2 -> "
LVALUET2.sin = LVALUET2.her
LVALUE2.code = ''
LVALUE2.addr = LVALUE2.addrher
NUMEXPRESSION -> TERM NUMEXPRESSION1
NUMEXPRESSION1.her = TERM.sin
NUMEXPRESSION.sin = NUMEXPRESSION1.node
NUMEXPRESSION.code = TERM.code || NUMEXPRESSION1.code
NUMEXPRESSION1.addrher = TERM.addr
NUMEXPRESSION.addr = NUMEXPRESSION1.addr

NUMEXPRESSION1 -> SUMMINUS NUMEXPRESSION
NUMEXPRESSION1.node = new node (SUMMINUS.node, NUMEXPRESSION1.her, NUMEXPRESSION.node)
NUMEXPRESSION1.addr = newTemp()
NUMEXPRESSION1.code = NUMEXPRESSION.code || gen(NUMEXPRESSION1.addr '=' NUMEXPRESSION1.addrher SUMMINUS.addr NUMEXPRESSION.addr)
NUMEXPRESSION1 -> "
NUMEXPRESSION1.node = NUMEXPRESSION1.her

NUMEXPRESSION1.addr = NUMEXPRESSION1.addrher
NUMEXPRESSION1.code = ‘’
SUMMINUS -> +
SUMMINUS.node = new leaf (id ,+)
SUMMINUS.addr = +
SUMMINUS -> -
SUMMINUS.node = new leaf (id ,-)
SUMMINUS.addr = -
TERM -> UNARYEXPR TERM3
TERM3.her = UNARYEXPR.sin
TERM.sin = TERM3.sin
TERM3.addrher = UNARYEXPR.addr
TERM.code = UNARYEXPR.code || TERM3.code
TERM.addr = TERM3.addr
TERM2 -> MULDIVMOD UNARYEXPR TERM3
TERM2.node = new node (MULDIVMOD.node, TERM2.her, TERM3.sin)
TERM2.addr = newTemp()
TERM2.code = UNARYEXPR.code || gen(TERM2.addr ‘=’
TERM2.addrher MULDIVMOD.addr UNARYEXPR.addr) || TERM3.code
TERM3.addrher = TERM2.addr
TERM3 -> TERM2
TERM2.her = TERM3.her
TERM2.addrher = TERM3.addrher
TERM3.sin = TERM2.node
TERM3.addr = TERM2.addr
TERM3.code = TERM2.code

TERM3 -> "
TERM3.sin = TERM3.her
TERM3.addr = TERM3.addrher
TERM3.code = ‘’
MULDIVMOD -> %
MULDIVMOD.node = new leaf(id,%)

MULDIVMOD.addr = %
MULDIVMOD -> /
MULDIVMOD.node = new leaf(id,/)
MULDIVMOD.addr = /
MULDIVMOD -> *
MULDIVMOD.node = new leaf(id,*)
MULDIVMOD.addr = *
UNARYEXPR -> SUMMINUS FACTOR
UNARYEXPR.sin = new node(SUMMINUS.node, FACTOR.node)
UNARYEXPR.addr = newTemp();
UNARYEXPR.code = FACTOR.code || gen(UNARYEXPR.addr '=' SUMMINUS.addr FACTOR.addr)//ou seja temp = +/- factor
UNARYEXPR -> FACTOR
UNARYEXPR.sin = FACTOR.node
UNARYEXPR.addr = FACTOR.addr
UNARYEXPR.code = FACTOR.code
FACTOR -> int-constant
FACTOR.node = new leaf(id, tabSimbolo(int-constant))
FACTOR.addr = tabSimbolo(int-constant);
FACTOR.code = ''
FACTOR -> string-constant
FACTOR.node = new leaf(id, ptabSimbolo(string-constant))
FACTOR.addr = tabSimbolo(string-constant);
FACTOR.code = ''
FACTOR -> null
FACTOR.node = new leaf(id, tabSimbolo(null))
FACTOR.addr = tabSimbolo(null);
FACTOR.code = ''
FACTOR -> LVALUE
FACTOR.node = LVALUE.node
FACTOR.addr = LVALUE.addr
FACTOR.code = LVALUE.code
FACTOR -> (NUMEXPRESSION)

FACTOR.node = NUMEXPRESSION.sin
FACTOR.addr = NUMEXPRESSION.addr
FACTOR.code = NUMEXPRESSION.code