UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

CAMPUS CORNÉLIO PROCÓPIO

ENGENHARIA DA COMPUTAÇÃO

RENAN VICENTIN FABRÃO

VINÍCIUS AGUIAR MORAES

DAVID PRIMO

RELATÓRIO DE ERROS

COMPILADOR TIGER

CORNÉLIO PROCÓPIO

2014

RENAN VICENTIN FABRÃO

VINÍCIUS AGUIAR MORAES

DAVID PRIMO

RELATÓRIO DE ERROS: COMPILADOR TIGER

Relatório do trabalho de correção de erros do compilador Tiger apresentada como requisito parcial da disciplina de Compiladores do curso de Engenharia da computação da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Andre Yoshiaki Kashiwabara.

CORNÉLIO PROCÓPIO

2014

LISTA DE ILUSTRAÇÕES

Figura 1 - Código Que Gera .end no Arquivo Assembly 7
Figura 2 - Código Corrigido (.end).....7
Figura 3 - Arquivo Assembly Com o Antigo Erro Gerado 8
Figura 4 - Parte do Código do Compilador Onde São Gerados Códigos Assembly de Comparação Errados 9
Figura 5 - Código do Compilador Com os Erros dos Comparadores Corrigidos.....9
Figura 6 - Código Responsavel por Gerar o Assembly Para a Operação de Subtração.....10
Figura 7 - Código Corrigido da Operação de Subtração 11
Figura 8 - Código que Gera o Erro do Jump 11
Figura 9 - Código Corrigido do Jump 11
Figura 10 - Exemplo de Código Errado Gerado Pelo Compilador.....13
Figura 11 - Código Com Erro Quando Não Existe ELSE 15
Figura 12 - Provável Código Onde Pode Ser Corrigido este Erro de Comparação de Tipos Diferentes 16

SUMÁRIO

1 INTRODUÇÃO 6
2 ERROS CORRIGIDOS 7
2.1 ERRO 1 7
2.2 ERRO 2 8
2.3 ERRO 3 10
2.4 ERRO 4 11
2.5 ERRO 5 12
3 ERROS ENCONTRADOS E NÃO SOLUCIONADOS 13
3.1 ERRO NÃO SOLUCIONADO 1 13
3.2 ERRO NÃO SOLUCIONADO 2 14
3.3 ERRO NÃO SOLUCIONADO 3 14
3.4 ERRO NÃO SOLUCIONADO 4 15
4 CONCLUSÃO 17

1 INTRODUÇÃO

Este trabalho tem como o objetivo desenvolvimento do conhecimento na área de compiladores, desenvolvendo e corrigindo um compilador para linguagem tiger.

A linguagem tiger é uma linguagem simples, de fácil didática muito utilizada para aprendizado e desenvolvimento de diversas competências na área da computação.

Na disciplina de compiladores, utilizamos a linguagem tiger como base para desenvolver um compilador como exemplo e aprendizado.

Para desenvolvimento deste compilador, utilizamos o ANTLR, uma api com uma linguagem própria utilizada para desenvolvimento de tokens, parses, e assim desenvolver a parte sintática de um compilador.

Após utilização desta api, utilizamos a linguagem Java, para desenvolvimento do restante do compilador, onde desenvolvemos o inicio da analise semântica. Após isso, foi disponibilizado um código de um compilador para tiger em Java, que também utilizou o ANTLR, desenvolvido para x86, onde havia vários erros na analise semântica e geração de código intermediário.

Para este trabalho foi proposto que houvesse a alteração da plataforma de execução do código, onde, ao invés de utilizar x86 fosse utilizado MIPS, além da correção de erros anteriores a modificação.

2 ERROS CORRIGIDOS

2.1 ERRO 1

Erro: *".end main"* no começo do arquivo assembly.

Descrição: Erro encontrado na linha 87 do arquivo 'Tiger.java' onde estava sendo acrescentado um *".end"* em lugar incorreto.

```
87          out.println(".end  " + f.frame.name);
88          debug.flush();
89
90
```

Figura 1 - Código Que Gera .end no Arquivo Assembly

Correção: Foi removido o *".end"* do 'out.println' ficando '*out.println(f.frame.name);*'

```
87          out.println(f.frame.name);
88          debug.flush();
89
90
```

Figura 2 - Código Corrigido (.end)

Neste erro, podemos observar a geração do código .end no inicio do arquivo assembly, sendo desnecessário o mesmo, e podendo causar erros de execução. Sendo assim, após analise do erro, verificamos a inserção deste código no arquivo 'Tiger.java', no método emitProc(), assim deletamos do compilador a geração deste código incorreto.

```
1  .end  main
2          .text
3  .globl main
4  main:
5  main_framesize=80
6  sub $sp,main_framesize
7  sw $ra,-4+main_framesize($sp)
8  sw $s0,-8+main_framesize($sp)
9  sw $s1,-12+main_framesize($sp)
10 sw $s2,-16+main_framesize($sp)
11 sw $s3,-20+main_framesize($sp)
12 sw $s4,-24+main_framesize($sp)
13 sw $s5,-28+main_framesize($sp)
14 sw $s6,-32+main_framesize($sp)
15 sw $s7,-36+main_framesize($sp)
16 sw $s8,-40+main_framesize($sp)
17
```

Figura 3 - Arquivo Assembly Com o Antigo Erro Gerado

2.2 ERRO 2

Erro: Erro na leitura do arquivo assembly quando havia algum comparador lógico como entrada.

Descrição: Na classe '*Codegen.java'* dentro do método *'munchStm(CJUMPs)'* a partir da linha 124, todos os comparadores (EQ, NE, LT, LE, GT, GE, ULE, UGT, ULE, UGE) estavam sendo chamados utilizando a sintaxe de processadores x86.

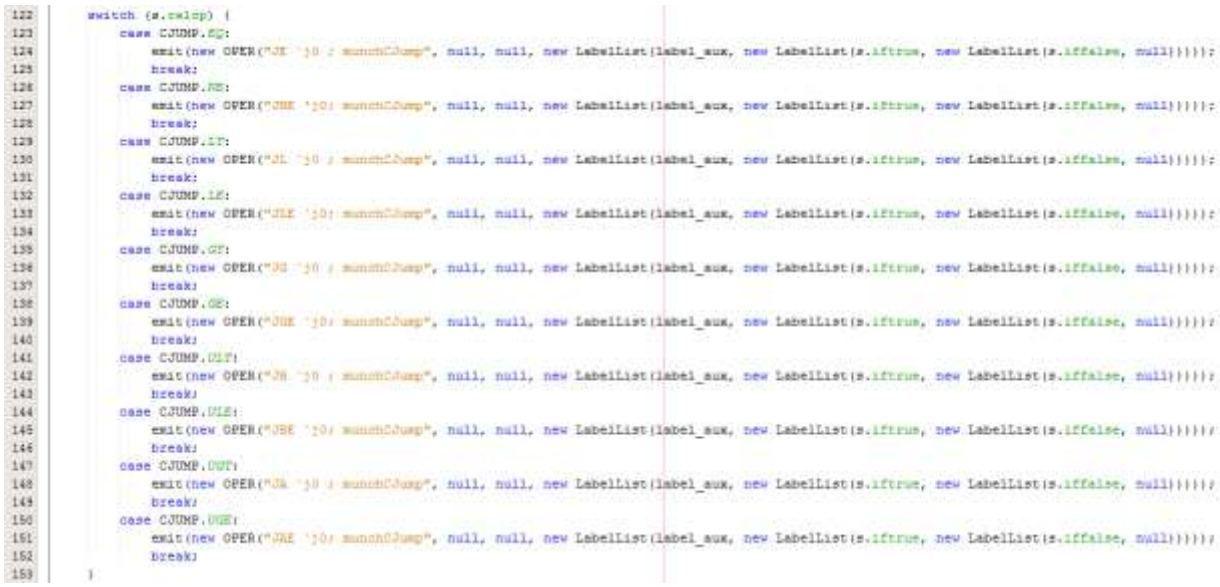

Figura 4 - Parte do Código do Compilador Onde São Gerados Códigos Assembly de Comparação Errados

Correção: Foi alterado para os comparadores equivalentes em linguagem MIPS.

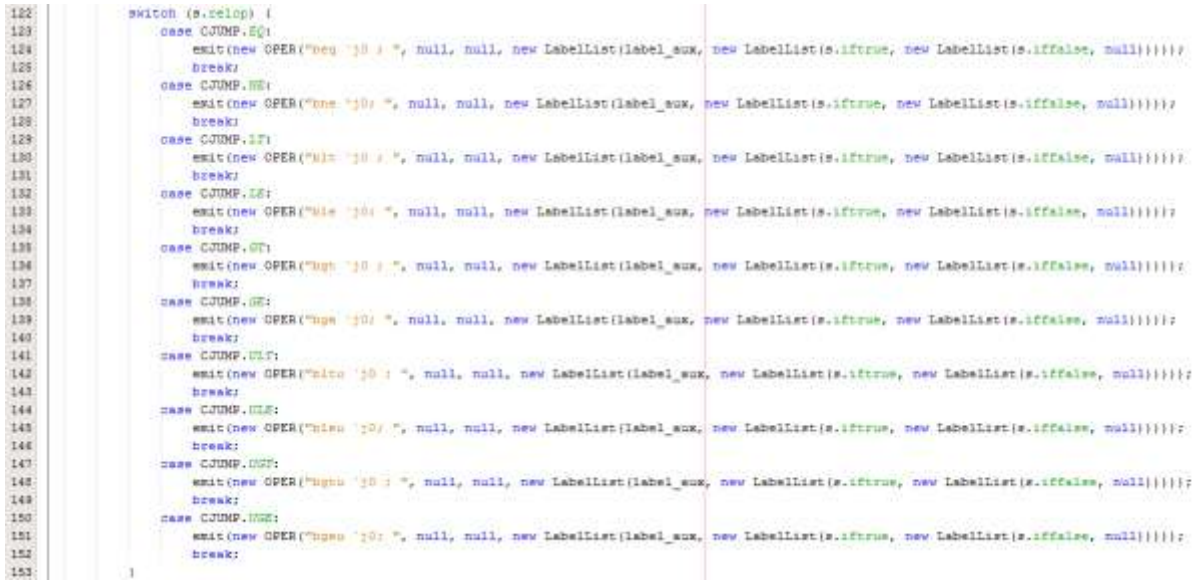

Figura 5 - Código do Compilador Com os Erros dos Comparadores Corrigidos

Neste erro, foi detectado a geração de códigos incompatíveis com o padrão assembly para MIPS, onde os códigos assembly gerados era para o padrão x86. Foi necessário uma pesquisa para identificação de códigos assembly que pudessem ser equivalentes aos anteriores, para arquitetura MIPS.

2.3 ERRO 3

Erro: Erro quando a entrada é uma operação de subtração.

Descrição: Na classe ‘*Codegen.java’* dentro do método *‘munchExp(BINOP e)’* na linha 295 e 299, o operador de subtração estava sendo chamado em linguagem x86.

```
300             if (e.binop == 1) {
301                 if (e.left instanceof CONST) {
302                     emit(OPER("subi `d0," + ((CONST) e.left).value + ",`s0", L(r),
303                             L(munchExp(e.right))));
304                     return r;
305                 }
306                 if (e.right instanceof CONST) {
307                     emit(OPER("subi `d0,`s0," + ((CONST) e.right).value + "", L(r),
308                             L(munchExp(e.left))));
309                     return r;
310                 }
311                 emit(OPER("sub `d0,`s0,`s1", L(r), L(munchExp(e.left),
312                         L(munchExp(e.right)))));
313                 return r;
314             }
315             emit(OPER(BINOP[e.binop] + " `d0,`s0,`s1", L(r), L(
316                     munchExp(e.left), L(munchExp(e.right)))));
317             return r;
```

Figura 6 - Código Responsavel por Gerar o Assembly Para a Operação de Subtração

Correção: Foi alterado de *‘subi’* para *‘subu’*, que é a operação equivalente ao ‘subi’ em MIPS.

```
300             if (e.binop == 1) {
301                 if (e.left instanceof CONST) {
302                     emit(OPER("subu `d0," + ((CONST) e.left).value, L(r),  L(munchExp(e.right))));
303                     return r;
304                 }
305                 if (e.right instanceof CONST) {
306                     emit(OPER("subu `d0,`s0," + ((CONST) e.right).value + "", L(r), L(munchExp(e.left))));
307                     return r;
308                 }
309                 //ELSE
310                 emit(OPER("sub `d0,`s0,`s1", L(r), L(munchExp(e.left), L(munchExp(e.right)))));
311                 return r;
312             }
313             emit(OPER(BINOP[e.binop] + " `d0,`s0,`s1", L(r), L(
314                     munchExp(e.left), L(munchExp(e.right)))));
315             return r;
```

Figura 7 - Código Corrigido da Operação de Subtração

Neste erro, observamos a geração de um código inexistente para assembly para Arquitetura MIPS, sendo assim foi necessário a realização de uma pesquisa para encontrar o comando equivalente ao ‘subi’, para a arquitetura MIPS.

2.4 ERRO 4

Erro: ‘Jump para false’, erro na leitura mips.

Descrição: Na classe *‘Codegen.java’* nas linhas 158 e 162 está sendo utilizado o comando *‘JMP’* é o comando de ‘*jump’* da linguagem x86.

```
156             emit(new OPER("JMP `j0 ; munchCJump", null, null, new LabelList(s.iffalse, null)));
157
158             /* faz o jump para false */
159             emit(new tiger.assem.LABEL(label_aux.toString() + ":", label_aux));
160             emit(new OPER("JMP `j0 ; munchCJump", null, null, new LabelList(s.iftrue, null)));
161
```

Figura 8 - Código que Gera o Erro do Jump

Correção: Foi substituido de ‘JMP’ para ‘j’, que é o equivalente em linguagem mips.

```
155             /* faz o jump para false */
156             emit(new OPER("j `j0 ;", null, null, new LabelList(s.iffalse, null)));
157
158             /* faz o jump para false */
159             emit(new tiger.assem.LABEL(label_aux.toString() + ":", label_aux));
160             emit(new OPER("j `j0 ;", null, null, new LabelList(s.iftrue, null)));
```

Figura 9 - Código Corrigido do Jump

Neste erro, observamos a geração de um código incompatível com o padrão para arquitetura MIPS em assembly, onde é gerado o código para x86.

Assim sendo, foi necessário a realização de uma pesquisa para encontrar um comando compatível com ‘JMP’, nesse caso ‘j’.

2.5 ERRO 5

Erro: Quando a entrada é algum comparador estava sendo gerado o arquivo assembly com o comando *‘munchCJump’* o qual não é reconhecido pelo mips.

Descrição: Na classe *‘Codegen.java’ n*as linhas 113,158,161 e 162 foram encontrados o comando *‘munchCJump’* o qual deveria ser apenas um comentário no código *assembly,* porém estava sendo gerado como um comando.

Correção: Foram removidos todos os *‘munchCJump’* do código.

Neste erro, observamos na geração do código assembly pelo compilador, onde em vários locais do arquivo ‘*Codegen.java*’, onde gerava um comando ‘*munchCJump*’, que não existe em nenhum dos padrões assembly. Acreditamos que esse comando deveria ser apenas um comentário. Na figura 4, existem vários exemplos de geração do código informado.

3 ERROS ENCONTRADOS E NÃO SOLUCIONADOS

3.1 ERRO NÃO SOLUCIONADO 1

Erro: Resultado final errado em operações de subtração.

Descrição: Quando a entrada é uma operação de subtração como por exemplo (1-2+3) o resultado está sendo 4 ao invés de 2, devido ao fato da arvore semântica esta sendo montada de forma incorreta assim fazendo com que o primeiro valor a ser passado para o registrador seja o segundo número da operação ficando (2-1+3).

```
18  L1:
19  addu $t0,$sp,main_framesize
20  sw $a0,0($t0)
21  addi $t0,$0,2
22  subu $t0,1
23  addi $t0,$t0,3
24  add $v0,$t0,$0
25  j L0
```

Figura 10 - Exemplo de Código Errado Gerado Pelo Compilador

Possível correção: Corrigir a forma com que a arvore semântica é montada.

Neste erro, foi possível observar a geração de código errado, quando é gerada uma subtração, invertendo os parâmetros, gerando resultados incoerentes com o esperado.

Realizamos uma primeira tentativa de correção deste erro, invertendo as ordens dos parâmetro ‘e.right’ e ‘e.left’, no código apresentado na figura 7. Nesta tentativa, conseguimos resolver um erro, quando a expressão utilizada contem apenas uma operação de subtração. Porém quando existe uma expressão com varias subtrações, é gerado um erro de construção da árvore sintática.

Para correção deste erro, seria necessário, a modificação da logica da construção da arvore semântica, pois na construção da arvore sintática esta sendo gerado o código correto, porem na tradução para código intermediário é onde ocorre este erro.

3.2 ERRO NÃO SOLUCIONADO 2

Erro: MIPS não consegue interpretar o arquivo assembly quando a entrada é ‘while’.

Descrição: Toda vez que a entrada for um ‘while’ é gerado o arquivo assembly com o comando CMP (linha 113 – Codegen) que é um comando de comparação utilizado no x86.

Neste erro foi possível observar a geração de um comando CMP, o qual é padrão x86. Após grande pesquisa realizada, não foi possível encontrar comando equivalente no padrão da arquitetura MIPS.

Nesta arquitetura também, os comandos de comparação, já informados anteriormente, podem realizar a comparação sem a necessidade do CMP assim como os próprios comandos gerarem o ‘jump’. Porém, devido a falta de conhecimento na lógica destes comandos, não foi possível gerar estes comandos corretamente.

3.3 ERRO NÃO SOLUCIONADO 3

Erro: Não está sendo gerado o arquivo assembly quando a entrada é uma estrutura ‘IF’ que não contém ‘ELSE’.

Descrição: Quando "elseclause", na classe ‘SemanticVisitor.java’, no método ‘visit(IfExp)’, instanciado como 'null' realiza a chamada dos métodos "elseclause.getTy()" e "elseclause.getExp()", porém, estes não retornam o valor esperado devida a "elseclause" ter sido instanciado como 'null'.

Possível correção: mudança na geração da arvore semântica do expressão ‘IF’.

```
366     @Override
        public void visit(IfExp e) {
368         e.test.accept(this);
369         ExpTy test = getExpTy();
370         checkInt(test, e.test.pos);
371         e.thenclause.accept(this);
372         ExpTy thenclause = getExpTy();
373         ExpTy elseclause = null;
374         if (e.elseclause != null) {
375             e.elseclause.accept(this);
376             elseclause = expTy;
377         }
            if (!thenclause.getTy().coerceTo(elseclause.getTy())
                    && (elseclause != null && !elseclause.getTy().coerceTo(thenclause.getTy()))) {
380         error(e.pos, "result type mismatch");
381         }
382         expTy = new ExpTy(getTranslate().IfExp(test.getExp(), thenclause.getExp(), elseclause.getExp()),
383                 elseclause.getTy());
384     }
```

Figura 11 - Código Com Erro Quando Não Existe ELSE

Obs: Quando a expressão de entrada houver "else" o código é gerado corretamente, porém sem a condição do IF, pois a instancia de "elseclause" é diferente de 'null'.

Neste erro foi observado, um erro na execução do compilador, quando existe uma expressão do tipo ‘IF’ que não contem um ‘ELSE’. Na analise semântica, quando executado visitor, é gerado um erro, pois a instancia ‘elseclause’, da classe ExpTy, é ‘null’, assim quando existe a chamada dos métodos utilizados por essa instancia, acontece erros.

Houve a tentativa de construir um novo método na classe ExpTy, o qual retornaria o resultado esperado, mesmo que a instanciação fosse ‘null’. Porem para gerar esse método seria necessário um grande aprofundamento nas classes do código como um todo, pois seria necessário, instanciar varias outras classes, pois devido a complexidade do compilador como um todo, há grande recursividade entre as classes, ficando assim inviável a correção deste erro.

3.4 ERRO NÃO SOLUCIONADO 4

Erro: O compilador está aceitando a comparação entre um inteiro e uma string.

Descrição: Por exemplo, quando temos na entrada estruturas como: *(for i:=10 to " " do i := i - 1)* e *(3 > "df")* o compilador deveria acusar uma mensagem de erro dizendo não ser possível tais comparações, porém no lugar da string está sendo alocado valores randômicos nos registradores.

Neste erro, quando executado um código como os exemplificados acima, o compilador deveria emitir uma mensagem de erro, informando que não é possível a comparação de tipos diferentes. Porém, ao invés disso, o compilador “transforma” a string, é uma representação de inteiro, e realiza a comparação destes valores.

```
434             switch (e.oper) {
435                 case OpExp.PLUS:
436                 case OpExp.MINUS:
437                 case OpExp.MUL:
438                 case OpExp.DIV:
439                     checkInt(left, e.left.pos);
440                     checkInt(right, e.right.pos);
441                     setExpTy(new ExpTy(getTranslate().OpExp(e.oper, left.getExp(), right.getExp()), INT));
442                     return;
443                 case OpExp.EQ:
444                 case OpExp.NE:
445                     checkComparable(left, e.left.pos);
446                     checkComparable(right, e.right.pos);
447                     if (STRING.coerceTo(left.getTy()) && STRING.coerceTo(right.getTy())) {
448                         setExpTy(new ExpTy(getTranslate().OpExp(e.oper, left.getExp(), right.getExp()), INT));
449                         return;
450                     } else if (!left.getTy().coerceTo(right.getTy()) && !right.getTy().coerceTo(left.getTy())) {
451                         error(e.pos, "incompatible operands to equality operator");
452                     }
453                     setExpTy(new ExpTy(getTranslate().OpExp(e.oper, left.getExp(), right.getExp()), INT));
454                     return;
455                 case OpExp.LT:
456                 case OpExp.LE:
457                 case OpExp.GT:
458                 case OpExp.GE:
459                     checkOrderable(left, e.left.pos);
460                     checkOrderable(right, e.right.pos);
461                     if (STRING.coerceTo(left.getTy()) && STRING.coerceTo(right.getTy())) {
462                         setExpTy(new ExpTy(getTranslate().StrOpExp(e.oper, left.getExp(), right.getExp()), INT));
```

Figura 12 - Provável Código Onde Pode Ser Corrigido este Erro de Comparação de Tipos Diferentes

Nesta parte de código do compilador, na classe ‘SemanticVisitor.java’, pode ser o local provável de correção do erro informado acima, pois há uma comparação entre os tipos, e onde é realizada a coerção dos mesmo.

4 CONCLUSÃO

Neste trabalho foi proposta correção de um compilador tiger que estava vários bugs utilizando o conhecimento adquirido no projeto anterior que foi desenvolvido pelos alunos. A correção dos bugs do compilador tiger representou um importante papel na compreensão da disciplina de Compiladores, onde pudemos nos concentrar no funcionamento do compilador como um todo, desde a parte léxica até a geração do código binário. Apesar das dificuldades encontradas na correção dos bugs devido a grande complexidade do projeto, os alunos conseguiram absorver grande conhecimento teórico e prático no conteúdo proposto pela disciplina.