

Análise e Transformação de Software

Bruno Filipe Martins Fernandes (Nº22827)

Luís Duarte Dias Machado (Nº22785)

Tiago Alves Carção (Nº23093)

Abril de 2013

Conteúdo

1	Introdução	2
2	Considerações Importantes	3
3	Gramática utilizada	4
4	Exemplo de programas	6
4.0.1	Exemplo 1	6
4.0.2	Exemplo 2	6
4.0.3	Exemplo 3	6
4.1	Árvores de prova	6
5	Tom	13
5.1	Estrutura de dados em Gom - Tom	13
5.2	Programa para gerar grafos e visualização textual	15
5.3	Árvores de prova do Tom dos exemplos	16
6	Nova implementação da Máquina Virtual MSP usando TOM+Java	20
6.1	Principais métodos	20
7	Geração de código para a Máquina Virtual	21
7.1	Principais métodos	21
8	Detecção de Falhas	24
8.1	Indicação de blocos de código executados pela nova Máquina Virtual	24
8.2	Formato dos ficheiros a fornecer à aplicação	24
8.3	Funcionamento da Aplicação	25
9	Geração de código MSP, utilização da nova máquina virtual, e detecção de falhas	26
9.1	Tradução do código	26
9.2	Funcionamento do software	27
10	Noção de função	30



11 Injecção de falhas em Software	30
11.1 Qualidade dos testes	30
11.2 Injecção de Falhas	31
11.3 Maus cheiros	33
11.4 Alterações à aplicação Java	34
11.5 Alterações à máquina virtual desenvolvida pelo grupo	36
12 Conclusão	37
13 Anexos	38
A Gramática para ANTLR v1	38
B Gramática para ANTLR com construtores Gom-Tom v1	43
C Gramática para ANTLR com construtores Gom-Tom v2	50
D Código Gom i- v2	51
E Código Main i- v2	53
F Gramática para ANTLR Máquina Virtual com construtores Gom-Tom	61
G Código Gom Máquina Virtual	64
H Código Main Máquina Virtual	65
I Código java da aplicação de detecção de falhas	75
J Código Main i- v3	85
K Código Gom Máquina Virtual v2	102
L Código Main Máquina Virtual v2	103
M Código java da aplicação de detecção de falhas e análise de injeção de falhas	114



1 Introdução

No âmbito da disciplina de Análise e Transformação de Software foi pedido aos alunos o desenvolvimento de uma gramática que especificasse uma linguagem.

Essa gramática deverá ser capaz de determinar a validade de um programa escrito numa linguagem **i - -**. No geral, essa linguagem é sintacticamente muito semelhante com a linguagem C e é composta por declarações (**int x**), expressões (**4*3**), ciclos (**while** e **for**), instruções (**if**, **calls** e **funções**) e comentários (**/*...*/**).

Para ser possível implementar estratégias genéricas é necessário converter a gramática obtida numa representação em **Tom**. Esta representação tem de ser extensiva ao ponto de suportar a gramática definida bem como manter a informação completa com menos entropia.

Foi também pedido que, através do TOM, se criasse código que poderia ser lido pela máquina virtual MSP. O grupo optou por criar uma nova máquina virtual utilizando o **TOM+Java**. Esta nova máquina virtual é muito semelhante à MSP e indica que pedaços de código foram executados, e assim através deste *output* e através dos resultados esperados para um determinado código, poderia-se proceder à detecção de falhas no código, através de uma outra aplicação desenvolvida pelo grupo. O algoritmo de detecção de falhas utilizado foi o *Sepctrum-based Fault Localization* (SFL).

À nossa linguagem foi também adicionada a noção de função, permitindo assim que se invoque funções dentro de funções. Foram também utilizadas várias técnicas de injeção de falhas em software de maneira a assegurar a qualidades dos testes ao software. A aplicação Java desenvolvida pelo grupo permite que se tenha a noção dos blocos de códigos que foram utilizados através da utilização de um *ranking* e permite a consulta dos blocos de código executados e sua probabilidade de erro fornecendo testes com falhas. Todos esses dados podem ser guardados num ficheiro pdf, para posterior consulta.



2 Considerações Importantes

Neste relatório, seguem as especificações formais da gramática de forma a ser o mais legível possível.

Contudo, para validação da gramática é utilizado o "ANTLR". Este, não permite recursividade à esquerda o que levou o grupo a ter que adaptar a gramática de modo a eliminar tais recursividades.

Na especificação formal da gramática também não foi colocada a possibilidade de a linguagem **i** — aceitar comentários em qualquer sítio (por uma questão de legibilidade da gramática), contudo na gramática utilizada no ANTLR (em anexo) essa possibilidade existe.



3 Gramática utilizada

A gramática utilizada para definir a linguagem criada encontra-se descrita a seguir.

O Símbolo Inicial é *Programa*, os símbolos Terminais são escritos só em maiúsculas (terminais variáveis) ou só em minúsculas (palavras-reservadas) ou entre apóstrofes (sinais de pontuação); os restantes (sempre começados por maiúsculas) serão os Símbolos Não Terminais.

$\langle IdTipo \rangle \longrightarrow \text{int} \mid \text{char} \mid \text{float} \mid \text{boolean}$

$\langle Tipo \rangle \longrightarrow \text{INT} \mid \text{CHAR} \mid \text{FLOAT} \mid \text{BOOLEAN}$

$\langle Programa \rangle \longrightarrow \langle Declaracao \rangle \text{' ; ' } \langle Programa \rangle$
 $\quad \quad \quad \mid \langle Funcao \rangle \langle Programa \rangle$
 $\quad \quad \quad \mid \varepsilon^1$

$\langle Declaracao \rangle \longrightarrow \langle IdTipo \rangle (\text{ID} \mid \langle Atribuicao \rangle) (\text{' ; ' } (\text{ID} \mid \langle Atribuicao \rangle))^*$

$\langle Funcao \rangle \longrightarrow (\langle IdTipo \rangle \mid \text{void}) \text{ID ' (' } \langle Argumentos \rangle ? \text{') ' } \langle BlocoCodigo \rangle$

$\langle Argumentos \rangle \longrightarrow \langle IdTipo \rangle \text{ID (' ; ' } \langle Argumentos \rangle) ?$

$\langle Inst \rangle \longrightarrow \langle If \rangle \mid \langle For \rangle \mid \langle While \rangle \mid \langle Return \rangle \mid \langle Call \rangle \text{' ; ' }$

$\langle If \rangle \longrightarrow \text{if ' (' } \langle Condicao \rangle \text{') ' } \langle BlocoCodigo \rangle \langle Else \rangle$

$\langle Else \rangle \longrightarrow \text{else } \langle BlocoCodigo \rangle$
 $\quad \quad \quad \mid \text{else } \langle If \rangle$
 $\quad \quad \quad \mid \varepsilon$

$\langle While \rangle \longrightarrow \text{while ' (' } \langle Condicao \rangle \text{') ' } \langle BlocoCodigo \rangle$

$\langle For \rangle \longrightarrow \text{for ' (' } \langle IdTipo \rangle ? \langle Atribuicao \rangle \text{' ; ' } \langle Condicao \rangle \text{' ; ' } \langle Exp \rangle \text{') ' } \langle BlocoCodigo \rangle$

$\langle Return \rangle \longrightarrow \text{return (ID } \mid \langle Tipo \rangle \mid \langle Call \rangle) \text{' ; ' }$

$\langle Call \rangle \longrightarrow \text{ID ' (' } \langle Parametros \rangle ? \text{') ' }$

¹Representa o vazio.



$\langle Parametros \rangle$	$\longrightarrow (ID \mid \langle Tipo \rangle) (; \mid \langle Parametros \rangle) ?$
$\langle BlocoCodigo \rangle$	$\longrightarrow \{ \langle Codigo \rangle \}$
$\langle Codigo \rangle$	$\longrightarrow \begin{array}{l} \langle Atribuicao \rangle ; \mid \langle Codigo \rangle \\ \mid \langle Declaracao \rangle ; \mid \langle Codigo \rangle \\ \mid \langle Inst \rangle \mid \langle Codigo \rangle \\ \mid \varepsilon \end{array}$
$\langle Condicao \rangle$	$\longrightarrow \langle Cond \rangle (\langle OpCondicao \rangle \langle Condicao \rangle) ^*$
$\langle Cond \rangle$	$\longrightarrow \langle Expressao \rangle \langle OpCondicao \rangle \langle Expressao \rangle$
$\langle OpCondicao \rangle$	$\longrightarrow '>' \mid '<' \mid '>=' \mid '<=' \mid '!= ' \mid '== ' \mid '&\&' \mid ' '$
$\langle Expressao \rangle$	$\longrightarrow \begin{array}{l} \langle Expressao \rangle \langle OpExpressaoNumerica \rangle \langle Expressao \rangle \\ \mid \langle Condicao \rangle '?' \langle Expressao \rangle ':' \langle Condicao \rangle \\ \mid (INT \mid ID) \% (INT \mid ID) \\ \mid \langle IncOp \rangle ID \\ \mid ID \langle IncOp \rangle \\ \mid \langle OperadorUnario \rangle ? (ID \mid INT \mid FLOAT \mid BOOLEAN) \end{array}$
$\langle OperadorUnario \rangle$	$\longrightarrow '+' \mid '-' \mid '!'$
$\langle OpExpressaoNumerica \rangle$	$\longrightarrow '+' \mid '-' \mid '/' \mid '*'$
$\langle IncOp \rangle$	$\longrightarrow \begin{array}{l} '++' \\ \mid '--' \end{array}$
$\langle Atribuicao \rangle$	$\longrightarrow \begin{array}{l} ID \langle OpAtribuicao \rangle \langle Expressao \rangle \\ \mid ID '=' CHAR \\ \mid ID '=' \langle Call \rangle \end{array}$
$\langle OpAtribuicao \rangle$	$\longrightarrow '=' \mid '*=' \mid '/=' \mid '+=' \mid '-='$
$\langle Comentario \rangle$	$\longrightarrow \begin{array}{l} '/*' TEXTO '*/' \\ \mid '// TEXTO \end{array}$



4 Exemplo de programas

4.0.1 Exemplo 1

```
/*definicao variavel global*/
char c = 'a';

int maior(int a, int b) {
    if (a>b) {
        return a;
    }
    else {
        return b;
    }
}
```

4.0.2 Exemplo 2

```
void dump() {
    for (int i = 0; i < 2 && a; i++){
        a = i > 0 ? true : false;
    }
}
```

4.0.3 Exemplo 3

```
boolean whatever() {
    if (a > 2) {
        while (a) {
            return qualquer(a,true);
        }
    }
    /*nao deve chegar aqui*/
    return false;
}
```

4.1 Árvores de prova

Figura 1 Árvore do 1º exemplo sem os comentários representados na árvore

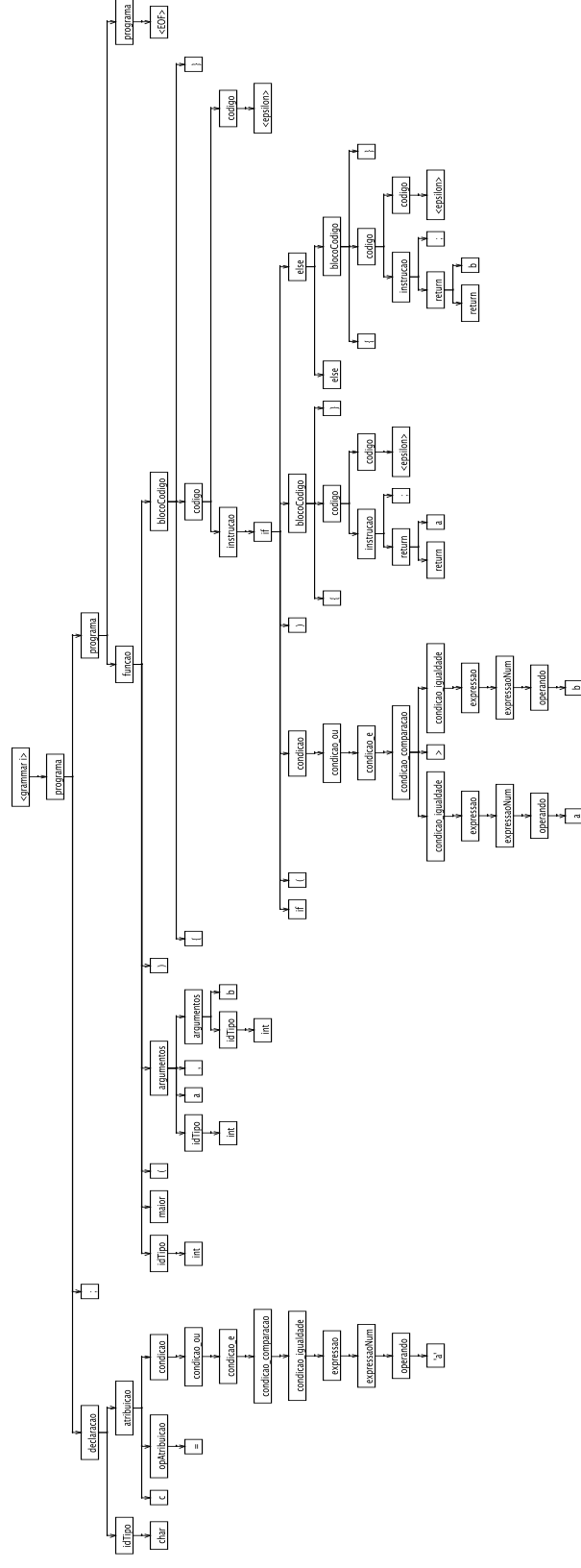


Figura 2 Árvore do 1º exemplo com os comentários representados na árvore

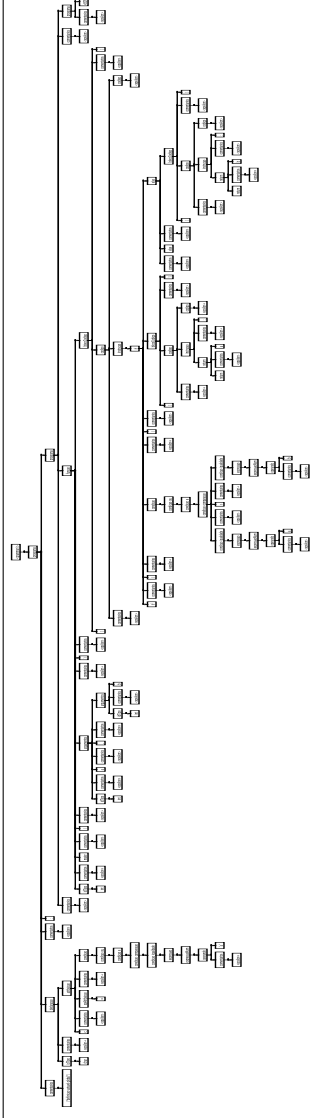


Figura 3 Árvore do 2º exemplo sem os comentários representados na árvore

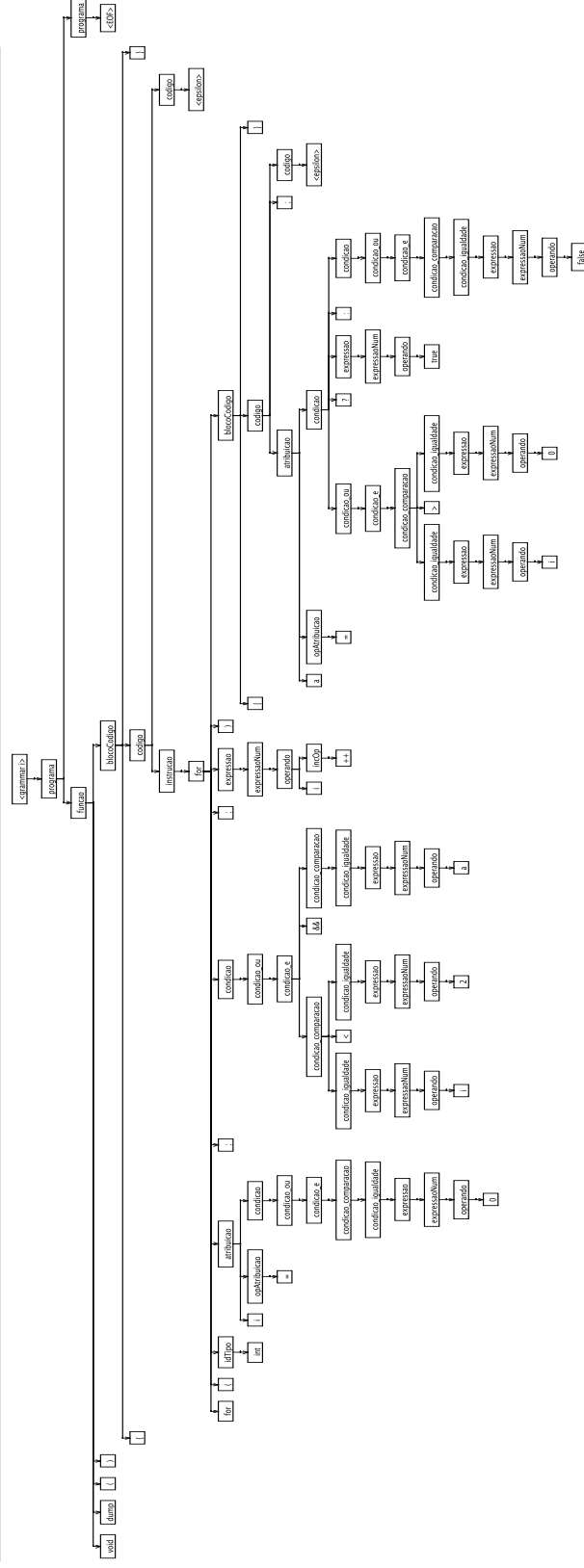
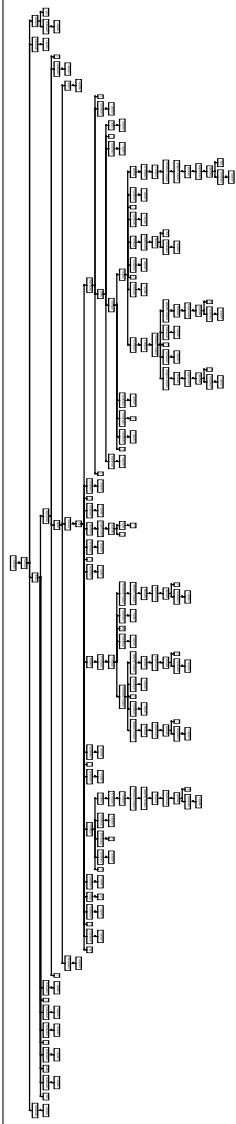


Figura 4 Árvore do 2º exemplo com os comentários representados na árvore



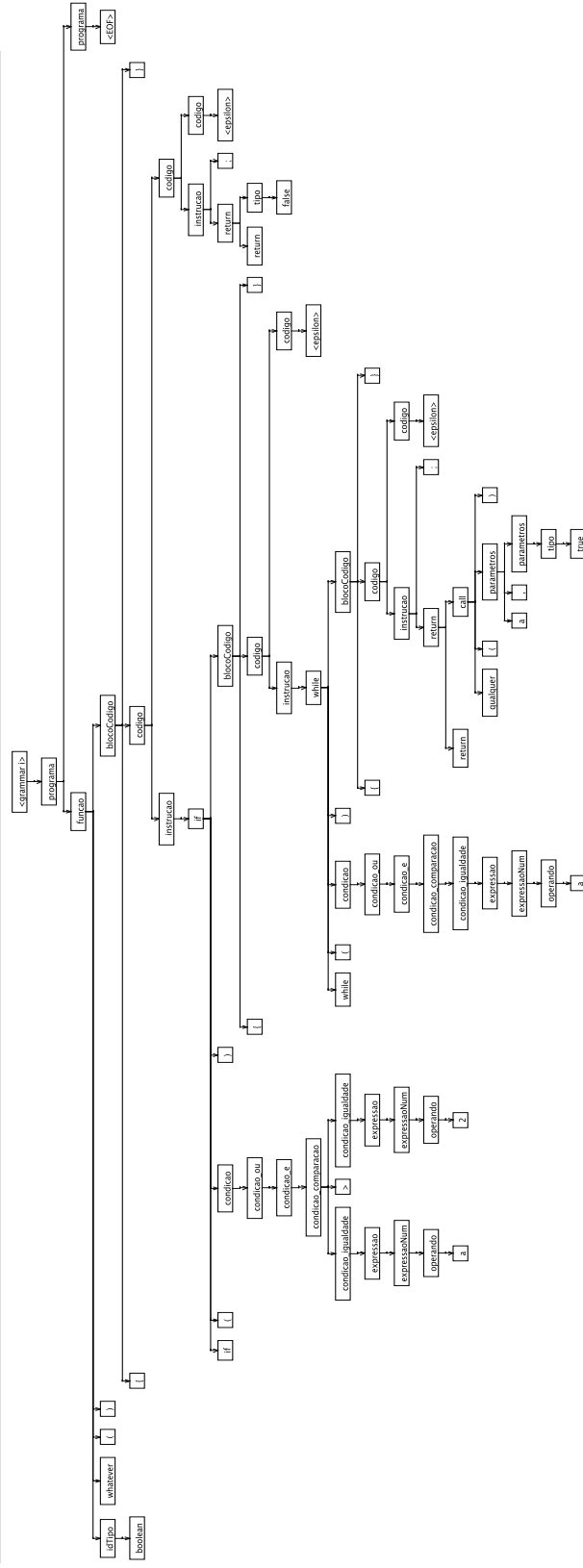
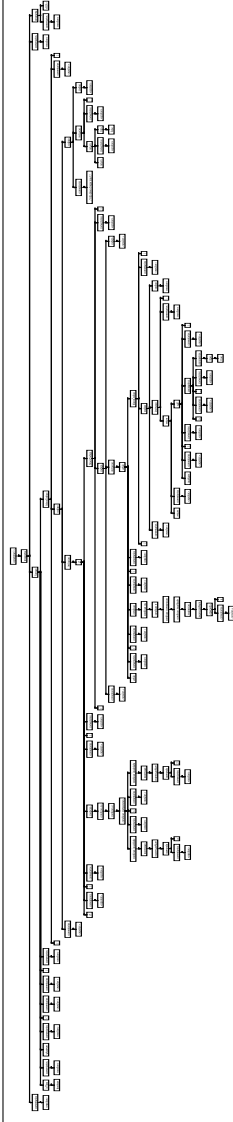


Figura 6 Árvore do 3º exemplo com os comentários representados na árvore





5 Tom

Para poder aplicar estratégias genéricas sobre a árvore da gramática, foi feita uma conversão da gramática do ANTLR para Tom. Para isso, foi definida uma estrutura de dados em Gom e depois as regras da gramática de ANTLR foram rescritas para utilizarem os construtores da estrutura do Gom.

A estrutura do Gom encontra-se definida a seguir, enquanto que a gramática de ANTLR com a utilização dos construtores encontra-se em anexo.

5.1 Estrutura de dados em Gom - Tom

```
module gram.i
imports int String
abstract syntax

Instrucao = Atribuicao(c1:LComentarios,Id:String,c2:LComentarios,op:
    OpAtribuicao,c3:LComentarios,Expressao:Expressao,c4:LComentarios)
    | Declaracao(c1:LComentarios,DefTipo:DefTipo,c2:LComentarios,
        Declaracoes:Declaracoes,c3:LComentarios,c4:LComentarios)
    | If(c1:LComentarios,c2:LComentarios,c3:LComentarios,Condicao:
        Expressao,c4:LComentarios,c5:LComentarios,Instrucao1:
        Instrucao,Instrucao2:Instrucao)
    | While(c1:LComentarios,c2:LComentarios,c3:LComentarios,
        Condicao:Expressao,c4:LComentarios,c5:LComentarios,
        Instrucao:Instrucao,c6:LComentarios)
    | For(c1:LComentarios,c2:LComentarios,Declaracao:Instrucao,c3:
        LComentarios,Condicao:Expressao,c4:LComentarios,c5:
        LComentarios,Expressao:Expressao,c6:LComentarios,c7:
        LComentarios,Instrucao:Instrucao,c8:LComentarios)
    | Return(c1:LComentarios,c2:LComentarios,Expressao:Expressao,
        c3:LComentarios)
    | Funcao(c1:LComentarios,DefTipo:DefTipo,c2:LComentarios,Nome:
        String,c3:LComentarios,c4:LComentarios,Argumentos:
        Argumentos,c5:LComentarios,c6:LComentarios,Instrucao:
        Instrucao,c7:LComentarios)
    | Exp(Expressao:Expressao)
    | SeqInstrucao(Instrucao*)

Expressao = ExpNum(Exp1:Expressao,c1:LComentarios,op:OpNum,c2:
    LComentarios,Exp2:Expressao)
    | Id(Id:String)
    | Pos(Expressao:Expressao)
    | Neg(Expressao:Expressao)
    | Nao(Expressao:Expressao)
    | Call(c1:LComentarios,Id:String,c2:LComentarios,c3:
        LComentarios,Parametros:Parametros,c4:LComentarios,c5:
        LComentarios)
    | IncAntes(OpInc:OpInc,Id:String)
    | IncDepois(OpInc:OpInc,Id:String)
```



```
| Condicional(Condicao:Expressao , c1:LComentarios , c2:
    LComentarios , Exp1:Expressao , c3:LComentarios , c4:
    LComentarios , Exp2:Expressao)
| Int(Int:int) | Char(Char:String) | True() | False() | Float
    (num:int)
| Ou(Cond1:Expressao , c1:LComentarios , c2:LComentarios , Cond2:
    Expressao)
| E(Cond1:Expressao , c1:LComentarios , c2:LComentarios , Cond2:
    Expressao)
| Comp(Exp1:Expressao , c1:LComentarios , OpComp:OpComp, c2:
    LComentarios , Exp2:Expressao)
| Empty()

DefTipo = DInt() | DChar() | DBoolean() | DFloat() | DVoid()

Argumentos = ListaArgumentos(Argumentos*)
    | Argumento(c1:LComentarios , DefTipo:DefTipo , c2:LComentarios ,
        Id:String , c3:LComentarios)

Parametros = ListaParametros(Parametros*)
    | Parametro(c1:LComentarios , Expressao:Expressao , c2:
        LComentarios)

Declaracoes = ListaDecl(Declaracoes*)
    | Decl(Id:String , c1:LComentarios , c2:LComentarios , Expressao:
        Expressao , c3:LComentarios)

OpAtribuicao = Atrib() | Mult() | Div() | Soma() | Sub()

OpNum = Mais() | Vezes() | Divide() | Menos() | Mod()

OpComp = Maior() | Menor() | MaiorQ() | MenorQ() | Dif() | Igual()

OpInc = Inc() | Dec()

LComentarios = Comentarios(LComentarios*)
    | Comentario(comentario:String)
    | Vazio()
```




```
    }  
    } catch (Exception e) {  
        System.err.println("exception:␣" + e);  
        return;  
    }  
}  
}
```

5.3 Árvores de prova do Tom dos exemplos

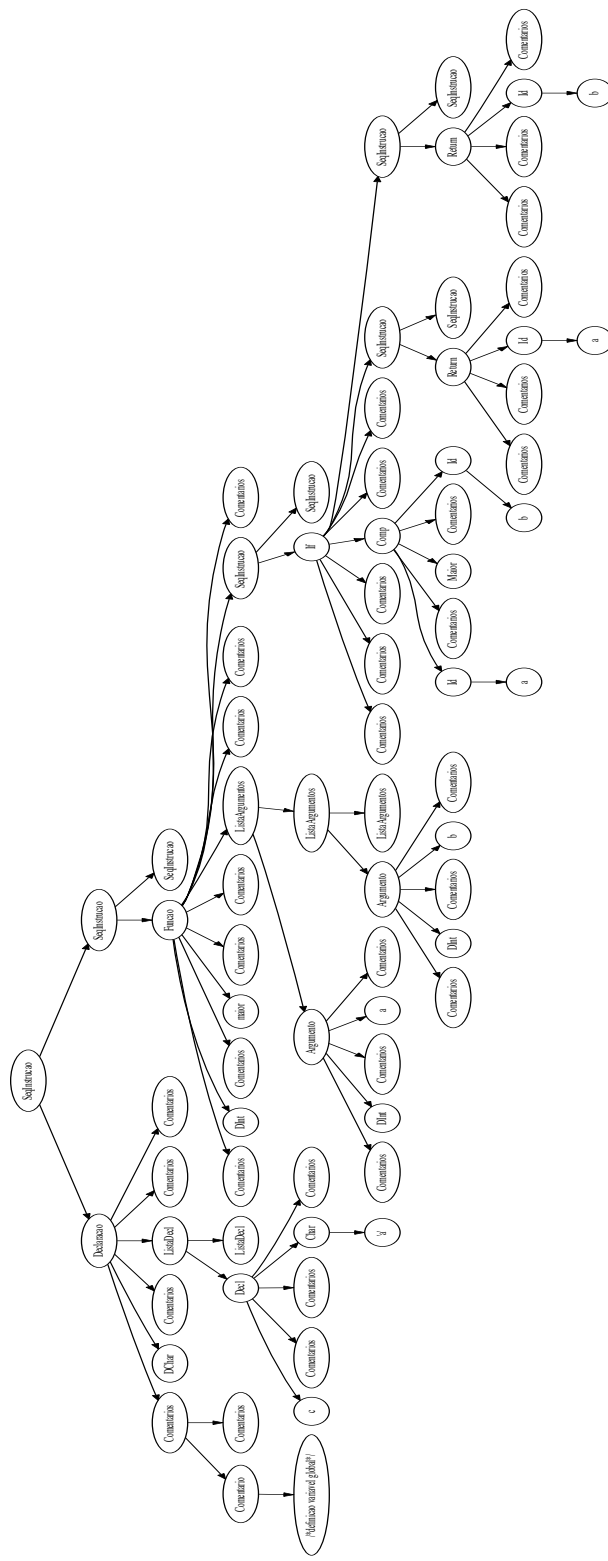


Figura 8 Árvore do 2º exemplo

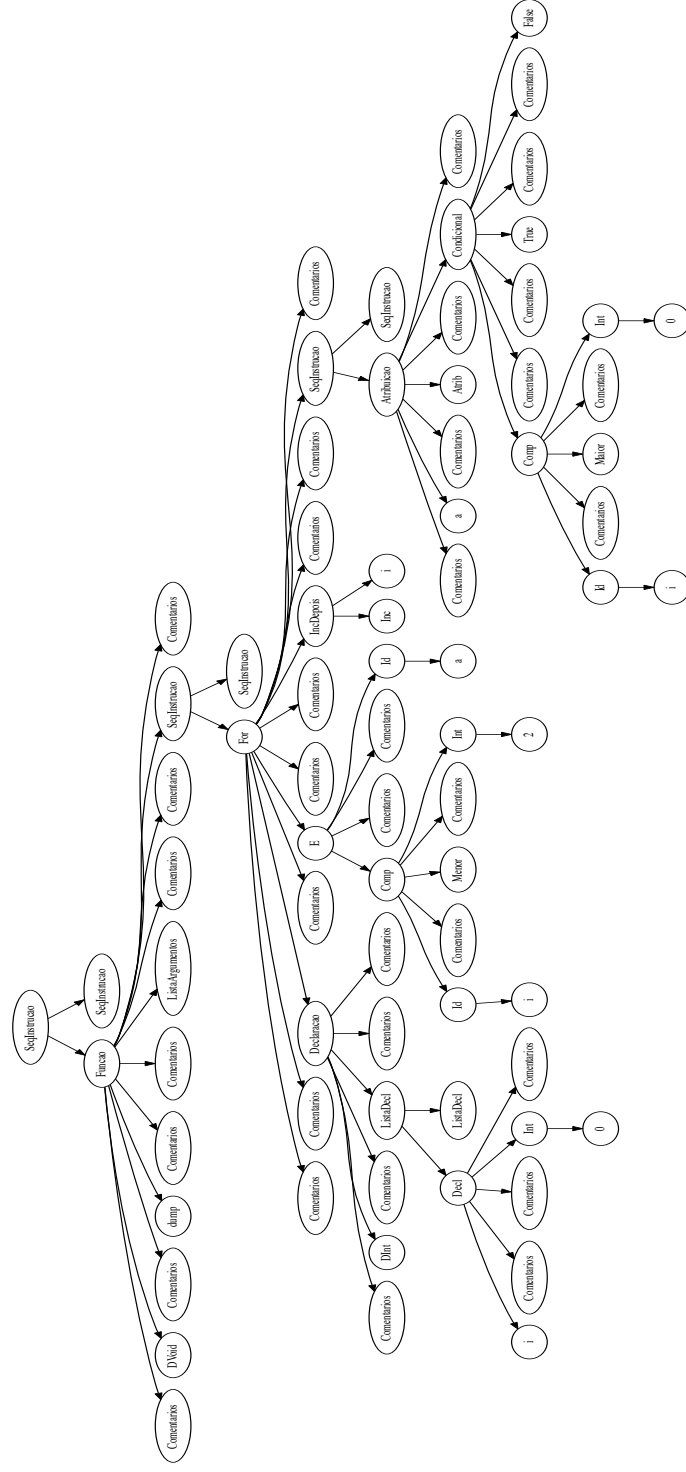
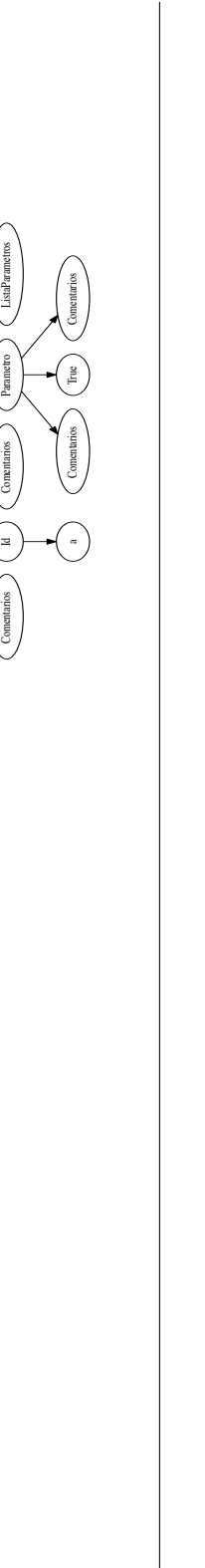
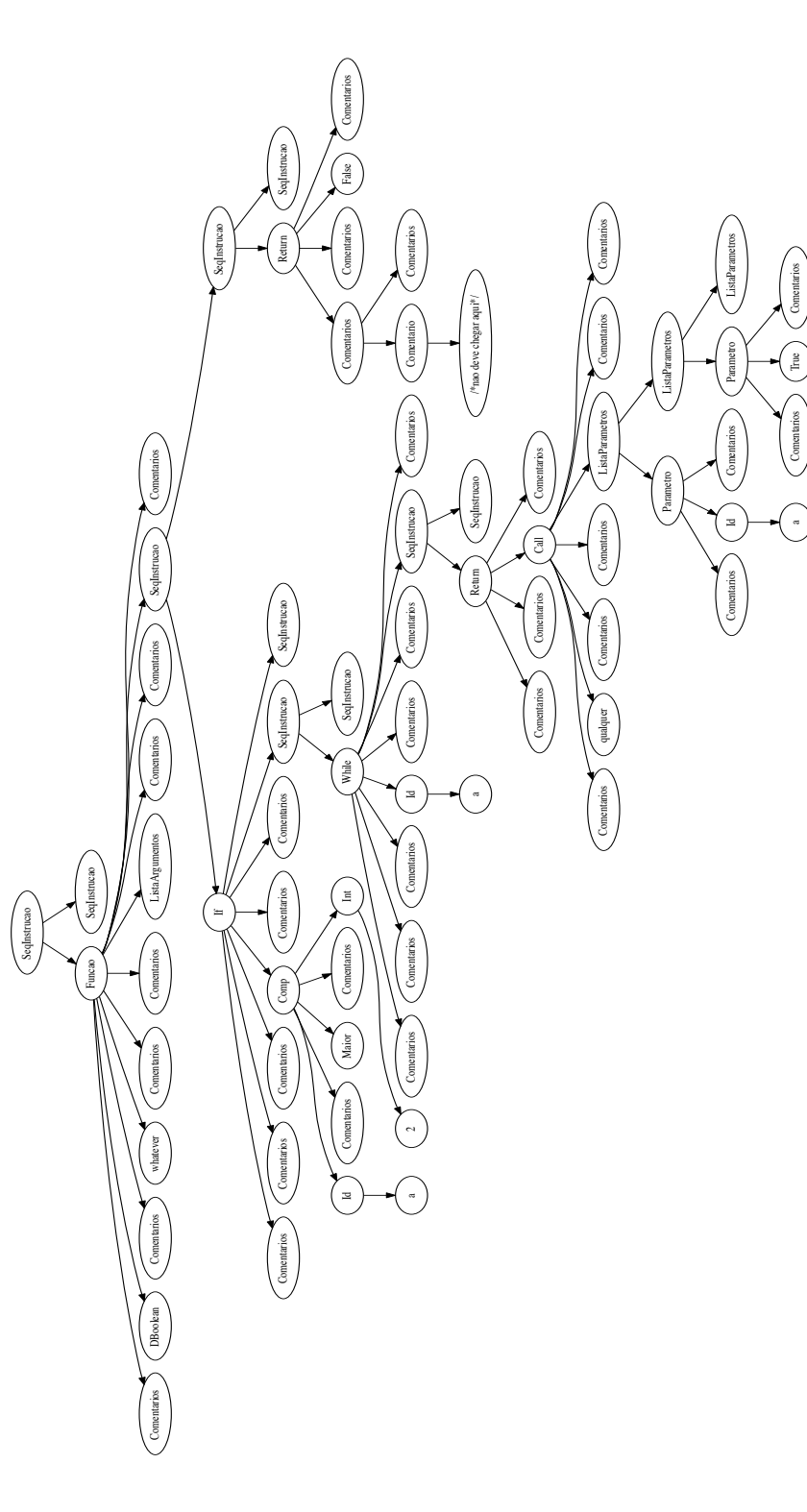


Figura 9 Árvore do 3º exemplo





6 Nova implementação da Máquina Virtual MSP usando TOM+Java

O grupo aceitou o desafio proposto pelo docente e desenvolveu uma nova máquina virtual usando **TOM+Java**. Esta nova máquina virtual é similar à máquina virtual MSP (Mais Simples Possível). Contudo, além de ter sido desenvolvida noutras tecnologias permite ao grupo entender a máquina virtual de maneira a torná-la mais completa no processamento da linguagem **i--** definida pelo grupo.

Devido à falta de tempo, o grupo não pode tornar a máquina virtual completamente compatível com todas as características da linguagem **i--**, ficando esse desafio como trabalho futuro. A nova máquina virtual possui as mesmas funcionalidades da MSP, sendo mais completa. Esta máquina virtual também fornece como output os blocos de código executados, para uma posterior detecção de falhas em software.

Em anexo, estão visíveis os ficheiros "msp.g", "msp.gom" e "Main.t" que correspondem à nova máquina virtual.

6.1 Principais métodos

A máquina virtual é composta por vários métodos definidos em Java. Existem métodos de controlo da "Stack" da máquina virtual:

Ficheiro 1 Excerto do ficheiro "Main.t". Métodos de controlo da stack.

```
private void pushStack(Termo termo){
    %match (stack){
        Stackk(termos*) -> { this.stack = 'Stackk(termo,termos*); }
    }
}

private void popStack(){
    %match (stack){
        Stackk(termo1,termos*) -> { this.stack = 'Stackk(termos*); }
    }
}

private Termo topStack() {
    %match(stack){
        Stackk(termo,termos*) -> { return 'termo; }
    }
    return 'Vazio();
}
```

Existe também o método run, que trata do funcionamento em si da máquina virtual. Neste método, define-se o comportamento da máquina virtual de acordo com as várias instruções existentes num programa. O código completo pode ser visto em anexo.



7 Geração de código para a Máquina Virtual

Após um estudo acerca do funcionamento da máquina virtual MSP (mais simples possível) e da criação da nova máquina virtual foi possível ao grupo começar a desenvolver métodos que fossem criando código para esta nova máquina virtual. Esta nova máquina virtual ainda não tem capacidade para tratar certas instruções da nossa linguagem *i*— que então não podem ser transformados em código legível pela máquina virtual.

Os métodos desenvolvidos pelo grupo encontram-se presentes no ficheiro *Main.t*, que se encontra em anexo. À medida que os métodos de geração do código iam sendo criados, algumas alterações mínimas iam sendo efectuadas nos ficheiros *.g* e *.gom*.

7.1 Principais métodos

Aqui são mostrados e explicados os principais métodos de criação de código legível pela MSP. Todo o código se encontra disponível em anexo.

As várias instruções e declarações existentes num programa devem ser traduzidos em código legível pela máquina virtual. Assim, de acordo com os termos existentes no ficheiro *"i.gom"* devem ser criados métodos (no ficheiro *"Main.t"*) que fazem um *"match"* entre um termo e o conjunto de termos do *gom*, de maneira a descobrir qual é o termo presente e aplicar o tratamento correcto.

Várias instruções e declarações devem ser codificados em código legível pela máquina virtual.

Como métodos de geração de código para a máquina virtual temos, entre outros, os métodos *"compileAnnotInstrucao"*, *"compileAnnotDeclaracoes"* e *"compileAnnotExpressoes"* que correspondem ao tratamento das instruções, declarações e expressões existentes na nossa linguagem *i*—, respectivamente. Como exemplo será mostrado um excerto do método *"compileAnnotExpressoes"*. Este método deve tratar todas as possibilidades de expressões existentes na nossa linguagem. Essas possibilidades encontram-se presentes no ficheiro *"i.gom"*. No excerto é possível verificar o tratamento das expressões *"ExpNum"*, *"Id"*, *"Pos"*, *"Neg"*, *"Nao"*, *"Call"*, *"IncAntes"* e *"Condicional"* que existem no ficheiro *"i.gom"*.



Ficheiro 2 Excerto do ficheiro "Main.t". Métodos de controlo da stack.

```
public String run(Instrucoes prog) {
    %match (prog){
        Instrucoes(inst,instrs*) -> {
            %match(inst) {
                ALabel(id) -> { return 'run(instrs*); }
                Call(id) -> {
                    'pushStack(I(pc));
                    prog = 'jmp(prog,id);
                    return 'run(prog);
                }
                Ret() -> {
                    Termo progCount = 'topStack();
                    %match(progCount) {
                        I(valor) -> { prog = 'getNInstr(prog,valor); }
                    }
                    return 'run(prog);
                }
                Add() -> {
                    %match (stack){
                        Stackk(I(v2),I(v1),resto*) -> {
                            stack = 'resto*;
                            int resultado = 'v1+'v2;
                            'pushStack(I(resultado));
                            return 'run(instrs*);
                        }
                    }
                }
                Sub() -> {
                    %match (stack){
                        Stackk(I(v2),I(v1),resto*) -> {
                            stack = 'resto*;
                            int resultado = 'v1 - 'v2;
                            'pushStack(I(resultado));
                            return 'run(instrs*);
                        }
                    }
                }
            }
            .....
        }
    }
}
```



Ficheiro 3 Excerto do ficheiro "Main.t" da linguagem desenvolvida pelo grupo. Excerto do método *compile Expressões with annotation* (compileAnnotExpressoes).

```
private String compileAnnotExpressoes(Expressao e, NumToInt numInstrucao) {
    %match(e) {
        ExpNum(exp1,_,op,_,exp2) -> {
            String genExp1 = 'compileAnnotExpressoes(exp1, numInstrucao);
            String genExp2 = 'compileAnnotExpressoes(exp2, numInstrucao);
            %match(op) {
                Mais() -> { return genExp1 + genExp2 + "Add,"
                    "+genAnnotation(numInstrucao.inc()); }
                Vezes() -> { return genExp1 + genExp2 + "Mul,"
                    +genAnnotation(numInstrucao.inc()); }
                Divide() -> { return genExp1 + genExp2 + "Div,"
                    +genAnnotation(numInstrucao.inc()); }
                Menos() -> { return genExp1 + genExp2 + "Sub,"
                    +genAnnotation(numInstrucao.inc()); }
                Mod() -> { return genExp1.concat(genExp2); }
            }
            return "";
        }
        Id(id) -> { return "Pushv \"" + 'id + "\",Load,"; }
        Pos(exp) -> { return 'compileAnnotExpressoes(exp, numInstrucao); }
        Neg(exp) -> { return 'compileAnnotExpressoes(exp, numInstrucao); }
        Nao(exp) -> {
            String genExp = 'compileAnnotExpressoes(exp, numInstrucao);
            return genExp + "Not";
        }
        Call(_,id,_,_,parametros,_,_) -> { return ""; }
        IncAntes(opInc,id) -> {
            %match(opInc) {
                Inc() -> { return "Pushv \"" + 'id + "\",Inc,Pushv \"" + 'id + "\",Load,"; }
                Dec() -> { return "Pushv \"" + 'id + "\",Dec,Pushv \"" + 'id + "\",Load,"; }
            }
            return 'id;
        }
        Condicional(condicao,_,_,exp1,_,_,exp2) -> {
            String genCondicao = 'compileAnnotExpressoes(condicao, numInstrucao);
            String genExp1 = 'compileAnnotExpressoes(exp1, numInstrucao);
            String genExp2 = 'compileAnnotExpressoes(exp2, numInstrucao);
            return genCondicao.concat(genExp1).concat(genExp2);
        }
        ...
    }
}
```



8 Detecção de Falhas

Para detecção de falhas no software, as tarefas efectuadas pelo grupo podem ser resumidas em, essencialmente, duas. Uma delas consiste em indicar, através do TOM, os pedaços de código que são processados para determinados inputs. Isso é feito no ficheiro *Main.t*. A segunda tarefa consiste na criação de uma aplicação, desenvolvida em Java utilizando o IDE NetBeans, que recebendo como input um ficheiros com dados de teste e outro ficheiro com o resultado esperado para esses mesmos testes, aplica o algoritmo *Spectrum-based Fault Localization* para a geração de um vector com a probabilidade de falhas dos blocos de código. É utilizado o *coeficiente de Jaccard* para calculo das similaridades e detecção do bloco causador do possível erro.

Na matriz gerada pela aplicação as linhas correspondem a casos de teste e as colunas correspondem a blocos de código. Quando uma coluna possui valor 1 para um determinado teste indica que esse bloco de código foi executado. Quando possui valor 0 indica que não foi executado. A última coluna corresponde ao vector de erros e indica se o resultado obtido está correcto ou errado (de acordo com o input correspondente aos resultados esperados). Se numa linha da última coluna estiver o valor 1 indica que ocorreu um erro para o teste correspondente a essa linha. Se estiver 0, tudo está correcto. Com estes dados é possível calcular a probabilidade de erro de uma coluna (bloco de código). Para isso utiliza-se o *coeficiente de Jaccard* para o cálculo de similaridades, segundo a fórmula:

$$s_j = \frac{n_{11}}{n_{11} + n_{10} + n_{01}}$$

onde "n11" corresponde ao número de vezes que acontece valor 1 tanto na coluna "j" como na coluna de erros, "n10" corresponde ao número de vezes que acontece o valor 1 na coluna "j" e o valor 0 na coluna de erros e "n01" corresponde ao valor 0 na coluna "j" e ao valor 1 na coluna de erros.

8.1 Indicação de blocos de código executados pela nova Máquina Virtual

A indicação dos blocos de código que são executados é da responsabilidade da máquina virtual desenvolvida pelo grupo. Para gerar um ficheiro com indicação dos blocos de código que são executados pela aplicação, os vários "System.out" vão sendo acumulados numa variável de nome "output". No final da execução do programa, se o utilizador tiver fornecido o parâmetro com o nome do ficheiro onde guardar o output do programa executado, então será guardado o output nesse mesmo ficheiro. Caso o utilizador não forneça o nome do ficheiro, o output será enviado para o "System.out". Isto é verificável no ficheiro "Main.t" da máquina virtual desenvolvida pelo grupo, que se encontra em anexo.

8.2 Formato dos ficheiros a fornecer à aplicação

O formato dos documentos a fornecer à aplicação deve seguir as seguintes normas:
O ficheiro com os resultados esperados deve possuir um resultado por cada linha.

Ficheiro 4 Excerto do ficheiro "res esp.txt" fornecido à aplicação.

12
1
14
12



No ficheiro com os resultados dos testes, cada linha corresponde a um teste (significa que o número de linhas em ambos os ficheiros deve ser igual, i.e. devem ser fornecidos tantos resultados esperados como testes) e deve-se utilizar a “,” como separador dos blocos de código que são executados pelo teste.

O número de colunas da matriz corresponde ao maior valor existente em todas as linhas, excepto o último valor de cada linha, que corresponde ao resultado do teste e deve ser precedido de um “;”.

Ficheiro 5 Excerto do ficheiro “testes.txt” fornecido à aplicação. É possível verificar que, no excerto, o maior valor para as linhas corresponde ao 33, significando que pelo menos, deve existir um bloco de código com o número 33 (e seus precedentes, i.e. do 1 ao 33).

```
2,5,6,1,8,12,33,15,7;12
1,2,3,4,5,6,7,8,12,14,15,20;6
4,3,2,1,4,5,6,14,7;14
2,3,5;12
```

8.3 Funcionamento da Aplicação

Fornecendo à aplicação os dois ficheiros de dados, é possível efectuar a análise e gerar um vector com a probabilidade de falhas dos blocos de código. Utiliza-se o algoritmo SFL e o *coeficiente de Jaccard* para calculo das similaridades. Na matriz gerada pela aplicação, a linha correspondente à probabilidade de erro é preenchida por uma escala da cor vermelha. Blocos de código com probabilidade de erro maior estão “pintados” com um vermelho mais escuro, enquanto que blocos de código com probabilidade de erro menor estão “pintados” com um vermelho mais claro.

É também possível gerar também um relatório em pdf com a matriz, o vector de erro e o vector de probabilidades. Esse relatório será guardado no mesmo caminho onde se encontra o ficheiro com os resultados esperados, que é fornecido à aplicação. Só é possível gerar relatórios em pdf após se fornecer, à aplicação, os dois ficheiros como input (dados de teste e resultados esperados).

No capítulo seguinte será mostrado o funcionamento da aplicação Java.



9 Geração de código MSP, utilização da nova máquina virtual, e detecção de falhas

Após se fazer "make" (trata tanto da linguagem **i--** como da máquina virtual), para se gerar código legível pela nova máquina virtual, deve-se proceder da seguinte maneira:

1. `cd genI/`
2. `javac gram/Main.java`
3. `java gram.Main "nomeDoFicheiroASerGuardadoParaDepoisSerExecutadoNaMaquinaVirtual" < FicheiroComOPrograma`

Para utilização da nova máquina virtual deve-se proceder de forma análoga à utilização da máquina virtual MSP. Um utilizador que queira utilizar a nova máquina virtual deve executar os seguintes comandos:

1. `cd genMaqV/`
2. `javac maqv/Main.java`
3. `java maqv.Main "nomeDoFicheiroAGuardarOutput"`
4. caso existam inputs no código gerado para a máquina virtual vai ser pedido ao utilizador para inserir o valor. Caso queira fornecer directamente um ficheiro à máquina virtual já com o input, deve-se substituir o último passo (passo 3) por:
 - `java maqv.Main "nomeDoFicheiroAGuardarOutput" < FicheiroComOsInputs`

9.1 Tradução do código

Na geração do código da linguagem **i--** definida, é feito um mapeamento das instruções do programa em instruções da linguagem MSP que se encontra em anexo. A estas instruções são adicionados prints do número das instruções executadas. O conjunto destas instruções do MSP são mais tarde convertidas em termos GOM da linguagem MSP em TOM.

Como exemplo da geração de código desde a linguagem **i--** temos:

Traducoes 1 Atribuição de uma variável em **i--**

`aux = 3;`

Traducoes 2 Tradução em linguagem a ser interpretada pelo MSP sendo a instrução número 1

`Pushv "aux", Pushi 3, Store, Pushc ', ', IOut, Pushi 1, IOut`

Traducoes 3 Interpretação em gom da linguagem MSP

`Instrucoes(Pushv(S("aux")),Pushi(I(3)),Store(),Pushc(S(", "),IOut(),Pushi(I(1)),IOut())`

A restante linguagem **i--** é convertida em MSP e interpretada de modo semelhante, podendo ser consultada no código em anexo.



9.2 Funcionamento do software

Foram realizados vários testes para este software. O seguinte ficheiro com um programa escrito segundo a linguagem **i--**, foi utilizado para gerar código para a máquina virtual.

Ficheiro 6 Programa escrito segundo a linguagem **i--** e utilizado para geração de código para a máquina virtual (ficheiro teste.txt).

```
void a(){
    int aux;
    int i;

    aux = input(int);
    aux = 10+15*aux;
    if (aux > 5) {
        i = 5;
    }
    else {
        i = 6;
    }
    print(';');
    print(aux); //Resultado obtido
}
```

Realizando os 3 passos acima explicados foi gerado código legível pela máquina virtual.

Ficheiro 7 Código legível pela máquina virtual (ficheiro teste.msp), gerado a partir do programa existente no ficheiro 6.

```
Decl "aux" ,Decl "i" ,Pushv "aux",IIn int,Store,Pushi 1,IOut,Pushv "aux",
Pushi 10,Pushi 15,Pushv "aux",Load,Mul,Pushc ',' ,IOut,Pushi 2,IOut,
Add,Pushc ',' ,IOut,Pushi 3,IOut,Store,Pushc ',' ,IOut,Pushi 4,IOut,
Pushv "aux",Load,Pushi 5,Gt,Pushc ',' ,IOut,Pushi 5,IOut,Jumpf "senao8",
Pushv "i",Pushi 5,Store,Pushc ',' ,IOut,Pushi 6,IOut,Jump "fse8",
ALabel "senao8",Pushv "i",Pushi 6,Store,Pushc ',' ,IOut,Pushi 7,IOut,
ALabel "fse8",Pushc ',' ,IOut,Pushi 8,IOut,Pushc ';' ,IOut,Pushv "aux",Load,IOut,Halt
```

De seguida, pode-se executar este novo código na máquina virtual, seguindo os passos acima definidos. Ao executar, o utilizador poderá definir um ficheiro para guardar os blocos de código do programa (instruções) que são executados (poderá executar vários testes). Através desse ficheiro gerado pela máquina virtual (neste caso, com o valor de input 3 deu como resultado: 1,2,3,4,5,6,8,55) e usando um outro ficheiro com os resultados esperados para cada teste, o utilizador pode então usar a aplicação Java criada pelo grupo. Para a detecção e análise de falhas, após se iniciar a aplicação, o utilizador deverá carregar os ficheiros correctos. Para isso deve-se premir o menu "Carregar" e fornecer os resultados esperados e os dados de teste (Figura 10).

Após fornecer os ficheiros à aplicação, o utilizador já terá a possibilidade de premir o menu "Avaliar", que gera a matriz. Poderá também premir o menu "Carregar" e de seguida premir a opção "Gerar Relatório PDF" que gerará um ficheiro chamado "FaultLocalization.pdf" (Figura 12) com a matriz, na mesma directoria do ficheiro com os resultados esperados, fornecido à aplicação (Figura 11).

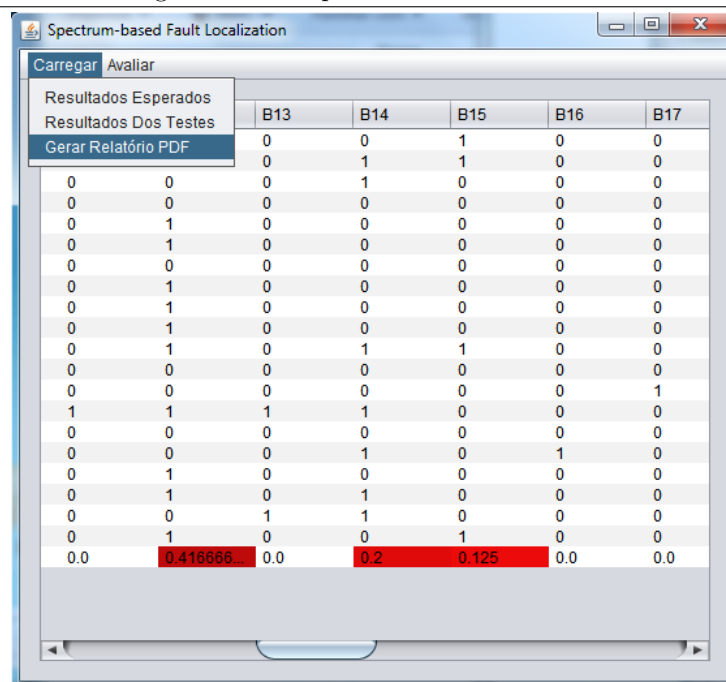
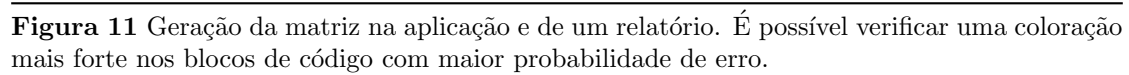
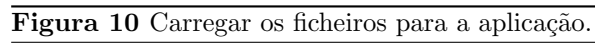




Figura 12 Uma página do relatório "FaultLocalization.pdf" gerado pela aplicação. O resto da matriz está presente nas restantes páginas, para melhor legibilidade.

FaultLocalization.pdf - Adobe Reader

Arquivo Editar Visualizar Janela Ajuda

1 / 6 66% Ferramentas Assinar Comentário

Spectrum-based Fault Localization

Este documento demonstra a matriz criada pela aplicação assim como o vector de erros.

1. Matriz

Teste	B1	B2	B3	B4	B5	B6
1	1	1	0	0	1	1
2	1	1	1	1	1	1
3	1	1	1	1	1	1
4	0	1	1	0	1	0
5	0	1	0	0	1	1
6	0	1	1	1	0	0
7	0	1	0	0	0	0
8	1	1	1	0	0	0
9	1	1	0	0	1	1
10	1	1	1	1	1	0
11	1	1	1	0	0	0
12	1	1	0	0	0	0
13	1	1	1	0	0	0
14	1	0	1	1	0	0
15	0	1	1	1	0	1
16	1	0	1	1	1	1
17	1	1	1	1	1	1
18	1	1	1	1	0	1
19	1	1	1	1	1	0
20	1	0	1	0	0	0
PE	0.1764706	0.2941176	0.25	0.3636363	0.25	0.2727272



10 Noção de função

Uma sempre importante funcionalidade que deve estar presente numa linguagem prende-se com a noção de função. Sendo assim, o grupo adicionou essa mesma noção tornando possível a invocação de várias funções dentro de outras funções. Esta noção permite maior liberdade ao utilizador na invocação de funções.

Para suportar esta característica na linguagem, foi necessário fazer uma travessia antes de gerar o código da linguagem para recolher o nome das funções e os seus argumentos.

11 Injecção de falhas em Software

Para se poder ter uma noção da qualidade do software é sempre necessário que se realizem vários testes ao software. Para isso, foi adicionada a possibilidade do utilizador injectar falhas no software através de um injector.

Se o utilizador quiser fornecer apenas como input algum programa deve executar o comando:

```
java gram.Main < PROGRAMA
```

Se quiser injectar falhas deve também fornecer o ficheiro com os inputs dos blocos de código que serão afectados com o formato "bloco,bloco,bloco". Um exemplo deste ficheiro será mostrado mais adiante.

Para beneficiar desta funcionalidade o utilizador deve executar o comando:

```
java gram.Main -fi INPUTSBLOCOS < PROGRAMA
```

Se quiser tratar dos "maus cheiros", o utilizador deve executar o comando:

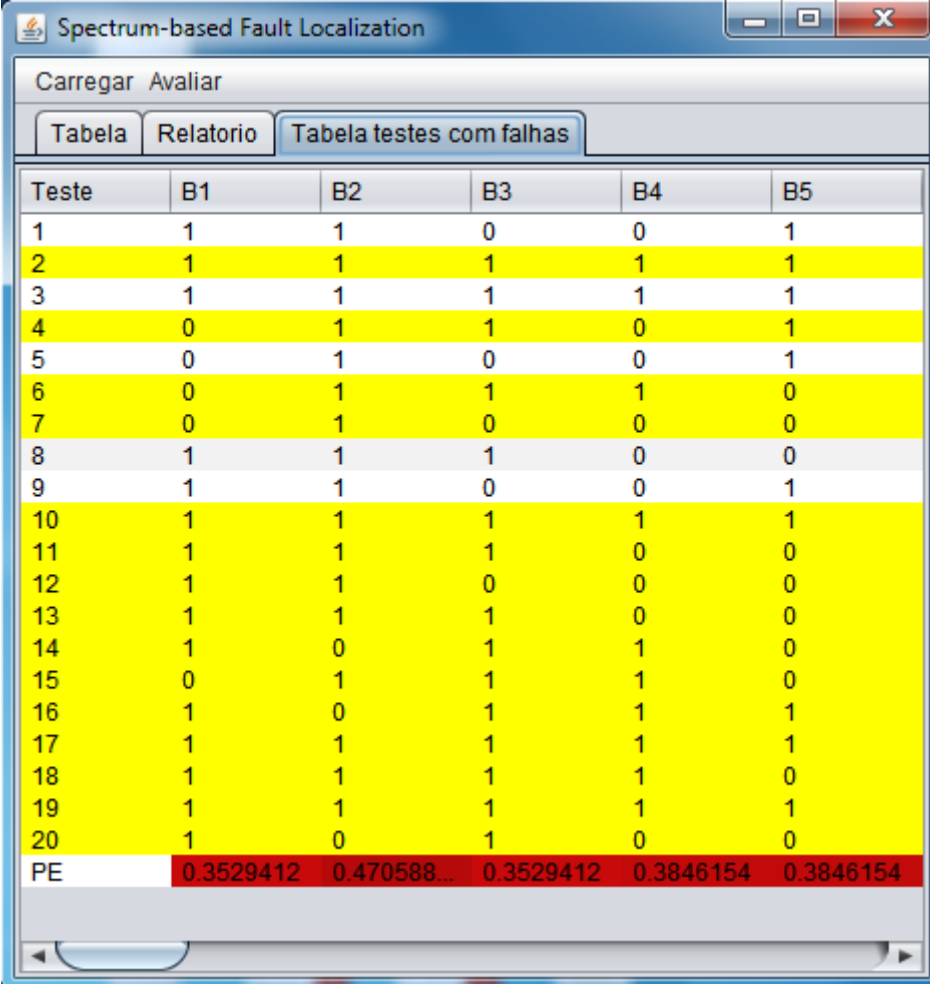
```
java gram.Main -bs < PROGRAMA
```

11.1 Qualidade dos testes

Através das funcionalidades adicionadas à aplicação java para detecção de falhas desenvolvida pelo grupo, o utilizador tem acesso a um ranking indicando que blocos de código são utilizados e quais é que não são e tem acesso a um ranking que mostra quais os blocos de código mais utilizados. Assim, através destas informações, o utilizador pode ter uma noção da qualidade dos seus testes. O utilizador pode utilizar estas informações para saber a cobertura dos seus testes (test coverage) através da informação relativa aos códigos que são, ou não, utilizados, podendo ter assim uma noção do código "morto" do seu software. Pode também ter indicações dos blocos de código cruciais para todo o bom funcionamento do software.



Figura 13 Matriz com blocos que não alteraram o estado coloridos, mesmo após a injeção de falhas



The screenshot shows a software window titled "Spectrum-based Fault Localization". It has a menu bar with "Carregar" and "Avaliar". Below the menu bar are three tabs: "Tabela", "Relatorio", and "Tabela testes com falhas", with the last one being selected. The main area contains a table with 6 columns: "Teste", "B1", "B2", "B3", "B4", and "B5". The table lists 20 tests (numbered 1 to 20) and a row labeled "PE". The data rows (1-20) have a yellow background, while the "PE" row has a red background. The values in the "PE" row are 0.3529412, 0.470588..., 0.3529412, 0.3846154, and 0.3846154.

Teste	B1	B2	B3	B4	B5
1	1	1	0	0	1
2	1	1	1	1	1
3	1	1	1	1	1
4	0	1	1	0	1
5	0	1	0	0	1
6	0	1	1	1	0
7	0	1	0	0	0
8	1	1	1	0	0
9	1	1	0	0	1
10	1	1	1	1	1
11	1	1	1	0	0
12	1	1	0	0	0
13	1	1	1	0	0
14	1	0	1	1	0
15	0	1	1	1	0
16	1	0	1	1	1
17	1	1	1	1	1
18	1	1	1	1	0
19	1	1	1	1	1
20	1	0	1	0	0
PE	0.3529412	0.470588...	0.3529412	0.3846154	0.3846154

11.2 Injeção de Falhas

A injeção de falhas no software é feita sobre três operações. Essas operações são o "while", "for" e o "if". O injetor de falhas troca as condições de teste efectuadas nessas operações. Se o ficheiro fornecido para injeção de falhas não referencia blocos de código passíveis de serem injectadas falhas então não são injectadas falhas e nada acontece. Esse ficheiro deve possuir os blocos de código mais utilizados (podem ser descobertos os blocos de código mais executados, através da aplicação Java desenvolvida pelo grupo. Isso será mostrado mais à frente neste relatório) de maneira a produzir informação relevante.

A injeção de falhas é feita utilizando estratégias em TOM. A estratégia aplicada altera a instrução mais utilizada desde que esta seja passível de ser alterada. Assim, a estratégia vai contando as instruções por onde passa, e quando atinge uma instrução que foi indicada pelo utilizador no ficheiro aplica a injeção da falha, caso contrário não faz nada.

Um exemplo da utilização do comando é:



```
java gram.Main -fi blocos.csv < ../fi.i
```

onde o ficheiro "blocos.csv" poderia ser composto apenas por:

3,7

A código da injeção de falhas foi o seguinte:

```
%strategy stratFaultInjectionWithKnowledge(ArrayList numInstrucao
,Set blocos) extends Identity() {
visit Instrucao {
    i@Atribuicao(_,_,_,_,_,_,_) -> {
        int num = (Integer) numInstrucao.remove((int) 0);
        numInstrucao.add(num+1);
        if (blocos.contains((Integer) num))
            return 'i;
    }
    If(c1,c2,c3,condicao,c4,c5,inst1,inst2) -> {
        int num = (Integer) numInstrucao.remove((int) 0);
        numInstrucao.add(num+1);
        if (blocos.contains((Integer) num))
            return 'If(c1,c2,c3,condicao,c4,c5,inst2,inst1);
    }
    While(c1,c2,c3,condicao,c4,c5,inst,c6) -> {
        int num = (Integer) numInstrucao.remove((int) 0);
        numInstrucao.add(num+1);
        if (blocos.contains((Integer) num))
            return 'While(c1,c2,c3,Nao(condicao),c4,c5,inst,c6);
    }
    For(c1,c2,decl,c3,condicao,c4,c5,exp,c6,c7,inst,c8) -> {
        int num = (Integer) numInstrucao.remove((int) 0);
        numInstrucao.add(num+1);
        if (blocos.contains((Integer) num))
            return 'For(c1,c2,decl,c3,Nao(condicao),c4,c5,exp,c6,
                c7,inst,c8);
    }
    i@Return(_,_,exp,_) -> {
        int num = (Integer) numInstrucao.remove((int) 0);
        numInstrucao.add(num+1);
    }
}
visit Expressao {
    e@ExpNum(exp1,_,_,op,_,_,exp2) -> {
        int num = (Integer) numInstrucao.remove((int) 0);
        numInstrucao.add(num+1);
    }
    e@Ou(cond1,_,_,cond2) -> {
        int num = (Integer) numInstrucao.remove((int) 0);
        numInstrucao.add(num+1);
    }
    e@E(cond1,_,_,cond2) -> {
```



```
        int num = (Integer) numInstrucao.remove((int) 0);
        numInstrucao.add(num+1);
    }
    e@Comp(exp1,_,opComp,_,exp2) -> {
        int num = (Integer) numInstrucao.remove((int) 0);
        numInstrucao.add(num+1);
    }
}
}
```

11.3 Maus cheiros

O grupo tratou também a problemática dos "maus cheiros" existentes em software. Os maus cheiros além de tornar o código menos legível podem também torna-lo menos eficiente. Para isso, tratar este problema é sempre uma boa solução para um software. Posto isto, o grupo trata essencialmente dois tipos de "maus cheiros", sendo eles:

- todas as variáveis existentes e que não são utilizadas são retiradas dos argumentos;
- e blocos de código onde a condição é a negação de um argumento passa a ser a validação do argumento de forma positiva. Isto é, por exemplo:

```
if(!x) { i = 1; }
else {i = 2;}
```

passa a

```
if(x) {i = 2;}
else {i = 1;}
```

O código utilizado para identificação e alteração dos maus cheiros é o seguinte:

```
public static Argumentos removeArgumentosNaoUtilizados(Argumentos
    args, TreeSet<String> idsUtilizados) {
    %match(args) {
        ListaArgumentos(arg1,tailArg*) -> {
            %match(arg1) {
                a@Argumento(_,_,_,idArg,_) -> {
                    if (idsUtilizados.contains('idArg'))
                        return 'ListaArgumentos(a,removeArgumentosNaoUtilizados
                            (tailArg*,idsUtilizados));
                    else
                        return removeArgumentosNaoUtilizados('tailArg*,
                            idsUtilizados);
                }
            }
        }
    }
    return args;
}
```



```
%strategy stratBadSmells() extends Identity() {
  visit Instrucao {
    If(c1,c2,c3,Nao(condicao),c4,c5,inst1,inst2) -> {
      return 'If(c1,c2,c3,condicao,c4,c5,inst2,inst1);
    }
    Funcao(c1, tipo ,c2,nome,c3,c4,argumentos,c5,c6,inst,c7) -> {
      TreeSet<String> idsUtilizados = new TreeSet<String>();
      'TopDown(stratCollectIds(idsUtilizados)).visit('inst);
      Argumentos args = removeArgumentosNaoUtilizados('argumentos
        ,idsUtilizados);
      return 'Funcao(c1, tipo ,c2,nome,c3,c4,args,c5,c6,inst,c7);
    }
  }
}

%strategy stratCollectIds(Set idsUtilizados) extends Identity() {
  visit Instrucao {
    Atribuicao(_,id,_,opAtrib,_,exp,_) -> {
      idsUtilizados.add('id);
    }
  }
  visit Expressao {
    Id(id) -> {
      idsUtilizados.add('id);
    }
    IncAntes(opInc,id) -> {
      idsUtilizados.add('id);
    }
    IncDepois(opInc,id) -> {
      idsUtilizados.add('id);
    }
  }
}
```

11.4 Alterações à aplicação Java

A aplicação desenvolvida pelo grupo na fase anterior sofreu algumas evoluções, principalmente de maneira a aceitar testes com falhas e assim poder-se proceder à comparação com os testes sem falhas. Foi também adicionada a possibilidade de visualização de um ranking com os blocos de código mais executados e ao ficheiro pdf que pode ser gerado pelo utilizador foi adicionada mais informação.

As novas funcionalidades podem ser visualizadas em anexo, na versão mais recente da aplicação Java.



Figura 14 Relatório da aplicação java sobre avaliação de testes

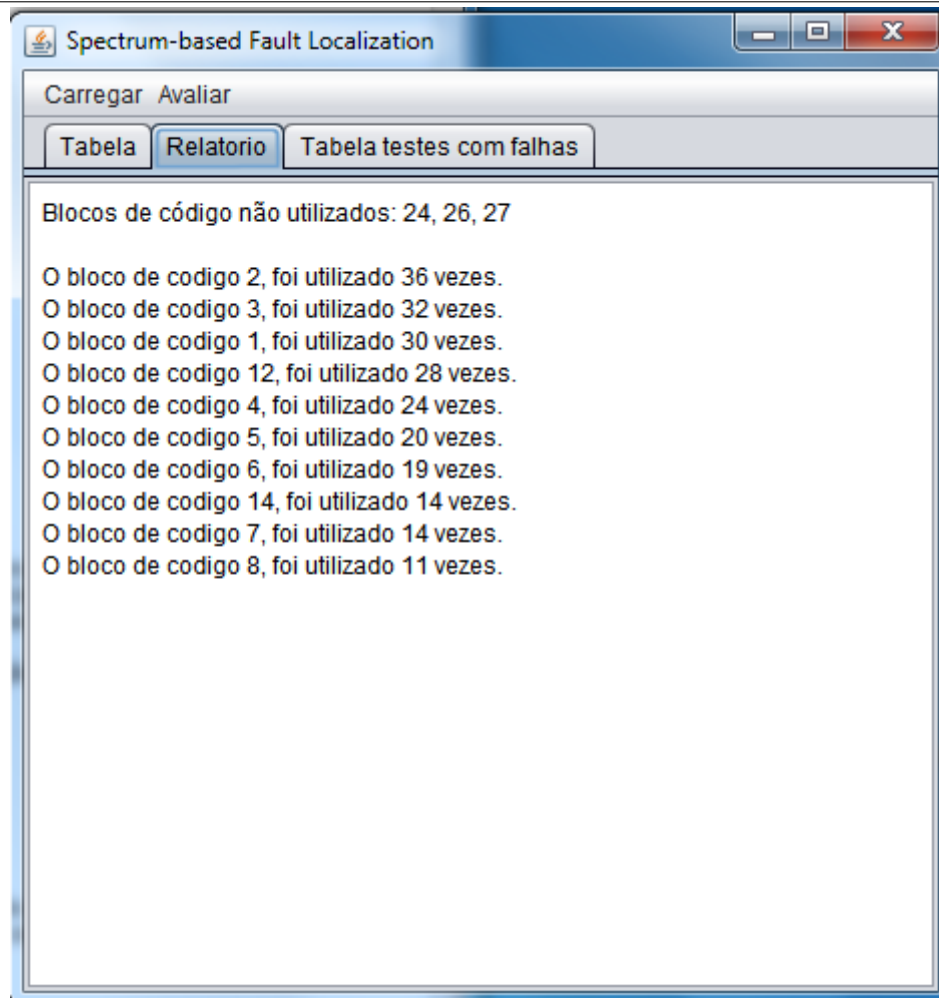




Figura 15 Menu de avaliação dos testes bem como dos testes com a injeção de falhas

Te:		B2	B3	B4	B5
1	1	1	0	0	1
2	1	1	1	1	1
3	1	1	1	1	1
4	0	1	1	0	1
5	0	1	0	0	1
6	0	1	1	1	0
7	0	1	0	0	0
8	1	1	1	0	0
9	1	1	0	0	1
10	1	1	1	1	1
11	1	1	1	0	0
12	1	1	0	0	0
13	1	1	1	0	0
14	1	0	1	1	0
15	0	1	1	1	0
16	1	0	1	1	1
17	1	1	1	1	1
18	1	1	1	1	0
19	1	1	1	1	1
20	1	0	1	0	0
PE	0.3529412	0.470588...	0.3529412	0.3846154	0.3846154

11.5 Alterações à máquina virtual desenvolvida pelo grupo

A máquina virtual desenvolvida pelo grupo sofreu algumas evoluções necessárias para continuar compatível com a linguagem i-. Sendo assim, a principal alteração à máquina virtual relaciona-se com a utilização de memória para endereçar variáveis. Esta nova funcionalidade é importante na medida em que a partir de agora esta máquina virtual permite a definição de arrays na linguagem i-.

As alterações efectuadas à máquina virtual podem ser visualizadas no anexo referente a versão mais recente da máquina virtual.

O grupo optou por injectar falhas no software não fazendo alterações na máquina virtual uma vez que a falta de tempo e a complexidade do injector não permitiram que esta funcionalidade fosse realizada em tempo útil.



12 Conclusão

Neste projecto, o grupo utilizou conhecimentos adquiridos em anteriores unidades curriculares bem como conhecimentos adquiridos na disciplina de Análise e Transformação de Software.

Inicialmente, o grupo começou por definir quais as instruções, expressões, etc. que estariam presentes na linguagem *i—*. De seguida, começou a especificar formalmente a gramática, de maneira a ir validando características dessa linguagem, começando pela definição de tipos, e de instruções, entre outros, até chegar aos ciclos e funções.

Até este momento o grupo não se preocupou com a possível existência de recursividade à esquerda, incompatíveis com o ANTLR.

Após terminar a especificação formal da gramática o grupo começou a escrever essa gramática no ANTLR. À medida que se ia passando produções para o ANTLR, foram sendo testadas essas mesmas produções de maneira a garantir que a gramática estava correcta. Todas as produções com recursividade à esquerda foram eliminadas e foi adicionada a possibilidade de existir comentários em praticamente todos os lugares.

Foi adaptada a gramática definida em ANTLR para poder utilizar as estratégias genéricas que o sistema Tom permite. A conversão teve em conta a utilização de menos identificadores para se tornar mais simples a manutenção e verificação de código.

O grupo adicionou a funcionalidade que permite a criação de código que é lido pela máquina virtual MSP. A máquina virtual MSP fornecida pelo professor, foi adaptada ao TOM, existindo assim a possibilidade de estender a máquina virtual de acordo com a linguagem *i—* definida pelo grupo.

Para a detecção de falhas foi desenvolvida uma nova aplicação que recebendo ficheiros de teste e ficheiros com os resultados esperados, indica qual o bloco de código mas susceptível a falhas.

Foi também adicionada a noção de função à nossa linguagem assim como foi tratada a noção de injeção de falhas em software. Foi adicionada a possibilidade de se fornecer à aplicação Java ficheiros de teste com falhas de maneira a avaliar os blocos de códigos que são utilizados e para se ter uma noção da qualidade dos testes utilizados.

Como trabalho futuro, os comentários serão utilizados para noções de qualidade e a gramática será estendida, de maneira a suportar *arrays*, mais tipos, entre outros.



13 Anexos

A Gramática para ANTLR v1

```
grammar i;

// definicao de tipos

idTipo  :      'char' | 'int' | 'boolean' | 'float' | 'void'
        ;

tipo    :      INT | FLOAT | CHAR | boolean_
        ;

// programa

prog :
    programa* EOF
    ;

programa :
    declaracao ';' | funcao
    ;

declaracao :
    comentarios idTipo comentarios dec_nodo ( comentarios ','
        comentarios dec_nodo )*
    ;

dec_nodo :
    ID comentarios ( '=' comentarios condicao comentarios )?
    ;

funcao :
    comentarios idTipo comentarios ID comentarios '(' comentarios
        argumentos? comentarios ')' comentarios blocoCodigo
        comentarios
    ;

argumentos :
    argumento ( ',' argumento )*
    ;

argumento :
    comentarios idTipo comentarios ID comentarios
    ;

// instrucoes
```



```
instrucao :
    if_ | for_ | while_ | return_ ';' | call ';'
    ;

if_ :
    comentarios 'if' comentarios '(' comentarios condicao
        comentarios ')' comentarios blocoCodigo ( else_ )?
    ;

else_ :
    'else' ( blocoCodigo | if_ )
    ;

for_ :
    comentarios 'for' comentarios '(' for_declaracao ';'
        comentarios condicao comentarios ';' comentarios expressao
        comentarios ')' comentarios blocoCodigo comentarios
    ;

for_declaracao :
    declaracao | atribuicao
    ;

while_ :
    comentarios 'while' comentarios '(' comentarios condicao
        comentarios ')' comentarios blocoCodigo comentarios
    ;

return_ :
    comentarios 'return' comentarios expressao comentarios
    ;

call :
    comentarios ID comentarios '(' comentarios parametros?
        comentarios ')' comentarios
    ;

parametros :
    parametro ( ',' parametro)*
    ;

parametro :
    comentarios expressao comentarios
    ;

blocoCodigo :
    '{' codigo* '}'
    ;
```




```
codigo :
    atribuicao ';' | declaracao ';' | instrucao
    ;

// expressao de condicao

condicao :
    condicao_ou ( comentarios '?' comentarios expressao
    comentarios ':' comentarios condicao )?
    ;

condicao_ou :
    (condicao_e -> condicao_e) ( comentarios '||' comentarios
    condicao_e )*
    ;

condicao_e :
    condicao_comparacao ( comentarios '&&' comentarios
    condicao_comparacao )*
    ;

condicao_comparacao :
    condicao_igualdade ( comentarios ('>' | '<' | '>=' | '<=')
    comentarios c=condicao_igualdade )*
    ;

condicao_igualdade :
    expressao ( comentarios ('!=' | '==') comentarios
    expressao )*
    ;

// expressao de atribuicao

atribuicao :
    comentarios ID comentarios opAtribuicao comentarios
    condicao comentarios
    ;

opAtribuicao :
    '=' | '*=' | '/=' | '+=' | '-='
    ;

// expressao numerica

expressao :
    expressaoNum ( comentarios ('+' | '-') comentarios expressaoNum
    )*
```



```

;

expressaoNum :
    oper ( comentarios ( '*' | '/' | '%' ) comentarios oper ) *
;

oper :
    ( opUnario ID
    | opUnario tipo
    | tipo
    | ID
    | incOp ID
    | ID incOp
    | call
    )
;

incOp :
    '++' | '--'
;

opUnario :
    '+' | '-' | '!'
;

boolean_ :
    'true' | 'false'
;

// comentario

comentarios :
    comentario *
;

comentario :
    ( COMENTARIO_LINHA | COMENTARIO_LINHAS )
;

COMENTARIO_LINHA
:      '//' ~ ( '\r' | '\n' ) *
;

COMENTARIO_LINHAS
:      '/*' ( options {greedy=false;} : . ) * '*/'
;

// Tokens lex
```



```
CHAR
:      '\'' ( '\\' ( 'b' | 't' | 'n' | 'f' | 'r' | '\"' | '\ ' | '\\ ' ) |
      ~ ( '\ ' | '\\ ' ) ) '\''
;

fragment DIGITO
:      ( '0' .. '9' )+
;

FLOAT
:      DIGITO+ '.' DIGITO* SuffixoFloat?
|      '.' DIGITO+ SuffixoFloat?
|      INT SuffixoFloat
;

SuffixoFloat
:      'f' | 'F'
;

INT
:      ( '0' | '1' .. '9' DIGITO* )
;

ID
:      LETRA ( LETRA | '0' .. '9' )*
;

fragment LETRA
:      'a' .. 'z' | 'A' .. 'Z' | '_'
;

WS : ( '\r' | '\t' | '\u000C' | '\n' ) { $channel=HIDDEN; }
;
```



B Gramática para ANTLR com construtores Gom-Tom v1

```
grammar i;

options {
    output=AST;
    ASTLabelType=Tree;
    tokenVocab=iTokens;
}

@header { package gram; }
@lexer::header { package gram; }

// definicao de tipos

idTipo :      ( 'char' -> ^(DChar) | 'int' -> ^(DInt) | 'boolean' ->
               ^(DBoolean) | 'float' -> ^(DFloat) | 'void' -> ^(DVoid) )
          ;

tipo :      (INT -> ^(Int INT) | FLOAT -> ^(Float FLOAT) | CHAR
             -> ^(Char CHAR) | boolean_ -> boolean_)
          ;

// programa

prog :
    programa* EOF -> ^(SeqInstrucao programa*)
    ;

programa :
    ( declaracao ';' -> declaracao
    | funcao -> funcao
    )
    ;

declaracao :
    c1=comentarios idTipo c2=comentarios dec_nodo ( c3=
        comentarios ',' c4=comentarios dec_nodo )* -> ^(Declaracao
        ^(Comentarios $c1?) idTipo ^(Comentarios $c2?) ^(
        ListaDecl dec_nodo*) ^(Comentarios $c3?) ^(Comentarios $c4
        ?))
    ;

dec_nodo :
    ( ID c1=comentarios -> ^(Decl ID ^(Comentarios $c1?) ^(
        Comentarios) Empty ^(Comentarios) )
    | ID c1=comentarios '=' c2=comentarios condicao c3=
        comentarios -> ^(Decl ID ^(Comentarios $c1?) ^(Comentarios
```



```
        $c2?) condicao ^(Comentarios $c3?))
    )
    ;

funcao :
    c1=comentarios idTipo c2=comentarios ID c3=comentarios '(' c4
        =comentarios argumentos? c5=comentarios ')' c6=comentarios
        blocoCodigo c7=comentarios -> ^(Funcao ^(Comentarios $c1
        ?) idTipo ^(Comentarios $c2?) ID ^(Comentarios $c3?) ^(
        Comentarios $c4?) ^(ListaArgumentos argumentos?) ^(
        Comentarios $c5?) ^(Comentarios $c6?) blocoCodigo ^(
        Comentarios $c7?))
    ;

argumentos :
    argumento ( ',' argumento )* -> argumento+
    ;

argumento :
    c1=comentarios idTipo c2=comentarios ID c3=comentarios -> ^(
        Argumento ^(Comentarios $c1?) idTipo ^(Comentarios $c2?)
        ID ^(Comentarios $c3?))
    ;

// instrucoes

instrucao :
    (if_ -> if_ | for_ -> for_ | while_ -> while_ | return_ ';'
        -> return_ | call ';' -> ^(Exp call))
    ;

if_ :
    c1=comentarios 'if' c2=comentarios '(' c3=comentarios
        condicao c4=comentarios ')' c5=comentarios blocoCodigo (
        else_ -> ^(If ^(Comentarios $c1?) ^(Comentarios $c2?) ^(
        Comentarios $c3?) condicao ^(Comentarios $c4?) ^(
        Comentarios $c5?) blocoCodigo else_)
    | -> ^(If ^(Comentarios $c1?) ^(Comentarios $c2?) ^(
        Comentarios $c3?) condicao ^(Comentarios $c4?) ^(
        Comentarios $c5?) blocoCodigo ^(SeqInstrucao) )
    )
    ;

else_ :
    'else' ( blocoCodigo -> blocoCodigo | if_ -> if_ )
    ;

for_ :
    ;
```



```
c1=comentarios 'for' c2=comentarios '(' for_declaracao ';' c3=
comentarios condicao c4=comentarios ';' c5=comentarios
expressao c6=comentarios ')' c7=comentarios blocoCodigo c8
=comentarios -> ^(For ^(Comentarios $c1?) ^(Comentarios
$c2?) for_declaracao ^(Comentarios $c3?) condicao ^(
Comentarios $c4?) ^(Comentarios $c5?) expressao ^(
Comentarios $c6?) ^(Comentarios $c7?) blocoCodigo ^(
Comentarios $c8?))
;

for_declaracao :
( declaracao -> declaracao
| atribuicao -> atribuicao
)
;

while_ :
c1=comentarios 'while' c2=comentarios '(' c3=comentarios
condicao c4=comentarios ')' c5=comentarios blocoCodigo c6=
comentarios -> ^(While ^(Comentarios $c1?) ^(Comentarios
$c2?) ^(Comentarios $c3?) condicao ^(Comentarios $c4?) ^(
Comentarios $c5?) blocoCodigo ^(Comentarios $c6?))
;

return_ :
c1=comentarios 'return' c2=comentarios expressao c3=
comentarios -> ^(Return ^(Comentarios $c1?) ^(Comentarios
$c2?) expressao ^(Comentarios $c3?))
;

call :
c1=comentarios ID c2=comentarios '(' c3=comentarios
parametros? c4=comentarios ')' c5=comentarios -> ^(Call ^(
Comentarios $c1?) ID ^(Comentarios $c2?) ^(Comentarios $c3
?) ^(ListaParametros parametros?) ^(Comentarios $c4?) ^(
Comentarios $c5?))
;

parametros :
parametro ( ',' parametro)* -> parametro+
;

parametro :
c1=comentarios expressao c2=comentarios -> ^(Parametro ^(
Comentarios $c1?) expressao ^(Comentarios $c2?))
;

blocoCodigo :
'{' codigo* '}' -> ^(SeqInstrucao codigo*)
```



```

;

codigo :
(
    atribuicao ';' -> atribuicao
|
    declaracao ';' -> declaracao
|
    instrucao -> instrucao
)
;

// expressao de condicao

condicao :
    condicao_ou ( c1=comentarios '?' c2=comentarios expressao c3=
        comentarios ':' c4=comentarios condicao -> ^(Condicional
            condicao_ou ^(Comentarios $c1?) ^(Comentarios $c2?)
            expressao ^(Comentarios $c3?) ^(Comentarios $c4?) condicao
        )
    | -> condicao_ou
    )
;

condicao_ou :
    (condicao_e -> condicao_e) ( c1=comentarios '||' c2=
        comentarios c=condicao_e -> ^(Ou $condicao_ou ^(
            Comentarios $c1?) ^(Comentarios $c2?) $c ) ) *
;

condicao_e :
    (condicao_comparacao -> condicao_comparacao) ( c1=comentarios
        '&&' c2=comentarios c=condicao_comparacao -> ^(E
            $condicao_e ^(Comentarios $c1?) ^(Comentarios $c2?) $c ) )
    *
;

condicao_comparacao :
    (condicao_igualdade -> condicao_igualdade) ( c1=
        comentarios ( '>' c2=comentarios c=
            condicao_igualdade -> ^(Comp $condicao_comparacao ^(
                Comentarios $c1?) ^(Maior) ^(Comentarios $c2?) $c )
    | '<' c2=comentarios c=condicao_igualdade -> ^(Comp
        $condicao_comparacao ^(Comentarios $c1?) ^(Menor) ^(
            Comentarios $c2?) $c )
    | '>=' c2=comentarios c=condicao_igualdade -> ^(Comp
        $condicao_comparacao ^(Comentarios $c1?) ^(MaiorQ) ^(
            Comentarios $c2?) $c )
    | '<=' c2=comentarios c=condicao_igualdade -> ^(Comp
        $condicao_comparacao ^(Comentarios $c1?) ^(MenorQ) ^(
            Comentarios $c2?) $c )
    )
;
```



```
)
)*
;

condicao_igualdade :
    (expressao -> expressao) ( c1=comentarios (
        '!= ' c2=comentarios e=expressao -> ^(Comp
            $condicao_igualdade ^(Comentarios $c1?) ^(Dif) ^(
                Comentarios $c2?) $e )
    | '==' c2=comentarios e=expressao -> ^(Comp
        $condicao_igualdade ^(Comentarios $c2?) ^(Igual) ^(
            Comentarios $c2?) $e )
    )
)*
;

// expressao de atribuicao

atribuicao :
    c1=comentarios ID c2=comentarios opAtribuicao c3=comentarios
    condicao c4=comentarios -> ^(Atribuicao ^(Comentarios $c1
        ?) ID ^(Comentarios $c2?) opAtribuicao ^(Comentarios $c3?)
        condicao ^(Comentarios $c4?))
    ;

opAtribuicao :
    ( '==' -> ^(Atrib)
    | '*==' -> ^(Mult)
    | '/==' -> ^(Div)
    | '+==' -> ^(Soma)
    | '-==' -> ^(Sub)
    )
    ;

// expressao numerica

expressao :
    (expressaoNum -> expressaoNum) ( c1=comentarios ( '+ '
        c2=comentarios e=expressaoNum -> ^(ExpNum $expressao ^(
            Comentarios $c1?) ^(Mais) ^(Comentarios $c2?) $e )
    | '- ' c2=comentarios e=expressaoNum -> ^(ExpNum $expressao ^(
        Comentarios $c1?) ^(Menos) ^(Comentarios $c2?) $e )
    )
)*
;

expressaoNum :
    (oper -> oper) ( c1=comentarios ( '*' c2=comentarios o=oper
        -> ^(ExpNum $expressaoNum ^(Comentarios $c1?) ^(Vezez) ^()
```




```
Comentarios $c2?) $o )
| '/' c2=comentarios o=oper -> ^(ExpNum $expressaoNum ^(
Comentarios $c1?) ^(Divide) ^(Comentarios $c2?) $o )
| '%' c2=comentarios o=oper -> ^(ExpNum $expressaoNum ^(
Comentarios $c1?) ^(Mod) ^(Comentarios $c2?) $o )
)
)*
;

oper :
( opUnario ID -> ^(opUnario ^(Id ID))
| opUnario tipo -> ^(opUnario tipo)
| tipo -> tipo
| ID -> ^(Id ID)
| incOp ID -> ^(IncAntes incOp ID)
| ID incOp -> ^(IncDepois incOp ID)
| call -> call
)
;

incOp :
( '++' -> ^(Inc) | '--' -> ^(Dec))
;

opUnario :
( '+' -> ^(Pos)
| '-' -> ^(Neg)
| '!' -> ^(Nao)
)
;

boolean_      :      ('true' -> ^(True) | 'false' -> ^(False))
;

// comentario

comentarios :
( comentario* -> comentario*
//      | -> ^(Vazio)
)
;

comentario :
( COMENTARIO_LINHA -> ^(Comentario COMENTARIO_LINHA)
| COMENTARIO_LINHAS -> ^(Comentario COMENTARIO_LINHAS)
)
;

COMENTARIO_LINHA
:      '//' ~('\'r' | '\n')*
```



```

;

COMENTARIO_LINHAS
:      '/'* ( options {greedy=false;} : . )* '*'/'
;

// Tokens lex

CHAR
:      '\'' ( '\\' ( 'b' | 't' | 'n' | 'f' | 'r' | '\"' | '\ ' | '\\ ' ) |
~( '\\ ' | '\\\ ' ) ) '\''
;

fragment DIGITO
:      ('0'..'9')+
;

FLOAT
:      DIGITO+ '.' DIGITO* SuffixoFloat?
|      '.' DIGITO+ SuffixoFloat?
|      INT SuffixoFloat
;

SuffixoFloat
:      'f' | 'F'
;

INT
:      ('0' | '1'..'9' DIGITO*)
;

ID
:      LETRA ( LETRA | '0'..'9' )*
;

fragment LETRA
:      'a'..'z' | 'A'..'Z' | '_'
;

WS : ( ' ' | '\r' | '\t' | '\u000C' | '\n' ) {$channel=HIDDEN;}
;
```



C Gramática para ANTLR com construtores Gom-Tom v2

Apenas alterações efectuadas.

```
call      :
    c1=comentarios ID  c2=comentarios '(' c3=comentarios
        parametros? c4=comentarios ')' c5=comentarios -> ^(Call ^(
            Comentarios $c1?) ID ^(Comentarios $c2?) ^(Comentarios $c3
                ?) ^(ListaParametros parametros?) ^(Comentarios $c4?) ^(
                    Comentarios $c5?))
    ;

print_    :
    c1=comentarios 'print' c2=comentarios '(' c3=comentarios
        expressao c4=comentarios ')' c5=comentarios -> ^(Print ^(
            Comentarios $c1?) ^(Comentarios $c2?) ^(Comentarios $c3?)
                expressao ^(Comentarios $c4?) ^(Comentarios $c5?))
    ;

oper      :
    ( opUnario ID -> ^(opUnario ^(Id ID))
    | opUnario tipo -> ^(opUnario tipo)
    | tipo -> tipo
    | ID -> ^(Id ID)
    | incOp ID -> ^(IncAntes incOp ID)
    | ID incOp -> ^(IncDepois incOp ID)
    | call -> call
    | input_ -> input_
    )
    ;
```



D Código Gom i– v2

Apenas alterações efectuadas.

```
Expressao = ExpNum(Exp1: Expressao , c1: LComentarios , op: OpNum, c2:
    LComentarios , Exp2: Expressao)
    | Id( Id: String )
    | Pos( Expressao: Expressao )
    | Neg( Expressao: Expressao )
    | Nao( Expressao: Expressao )
    | Call( c1: LComentarios , Id: String , c2: LComentarios , c3:
        LComentarios , Parametros: Parametros , c4: LComentarios , c5:
        LComentarios )
    | IncAntes( OpInc: OpInc , Id: String )
    | IncDepois( OpInc: OpInc , Id: String )
    | Condicional( Condicao: Expressao , c1: LComentarios , c2:
        LComentarios , Exp1: Expressao , c3: LComentarios , c4:
        LComentarios , Exp2: Expressao )
    | Int( Int: int ) | Char( Char: String ) | True() | False() | Float
        ( num: int )
    | Ou( Cond1: Expressao , c1: LComentarios , c2: LComentarios , Cond2:
        Expressao )
    | E( Cond1: Expressao , c1: LComentarios , c2: LComentarios , Cond2:
        Expressao )
    | Comp( Exp1: Expressao , c1: LComentarios , OpComp: OpComp , c2:
        LComentarios , Exp2: Expressao )
    | Input( c1: LComentarios , c2: LComentarios , c3: LComentarios , Tipo:
        DefTipo , c4: LComentarios , c5: LComentarios )
    | Empty()

Instrucao = Atribuicao( c1: LComentarios , Id: String , c2: LComentarios , op:
    OpAtribuicao , c3: LComentarios , Expressao: Expressao , c4: LComentarios )
    | Declaracao( c1: LComentarios , DefTipo: DefTipo , c2: LComentarios ,
        Declaracoes: Declaracoes , c3: LComentarios , c4: LComentarios )
    | If( c1: LComentarios , c2: LComentarios , c3: LComentarios , Condicao:
        Expressao , c4: LComentarios , c5: LComentarios , Instrucao1:
        Instrucao , Instrucao2: Instrucao )
    | While( c1: LComentarios , c2: LComentarios , c3: LComentarios ,
        Condicao: Expressao , c4: LComentarios , c5: LComentarios ,
        Instrucao: Instrucao , c6: LComentarios )
    | For( c1: LComentarios , c2: LComentarios , Declaracao: Instrucao , c3:
        LComentarios , Condicao: Expressao , c4: LComentarios , c5:
        LComentarios , Expressao: Expressao , c6: LComentarios , c7:
        LComentarios , Instrucao: Instrucao , c8: LComentarios )
    | Return( c1: LComentarios , c2: LComentarios , Expressao: Expressao ,
        c3: LComentarios )
    | Print( c1: LComentarios , c2: LComentarios , c3: LComentarios ,
        Expressao: Expressao , c4: LComentarios , c5: LComentarios )
    | Funcao( c1: LComentarios , DefTipo: DefTipo , c2: LComentarios , Nome:
        String , c3: LComentarios , c4: LComentarios , Argumentos:
```



```
Argumentos , c5 : LComentarios , c6 : LComentarios , Instrucao :  
Instrucao , c7 : LComentarios)  
| Exp(Expressao : Expressao)  
| SeqInstrucao(Instrucao*)
```



E Código Main i– v2

```
package gram;

import gram.i.iAdaptor;
import gram.i.types.*;
import org.antlr.runtime.CommonTokenStream;
import org.antlr.runtime.ANTLRInputStream;
import org.antlr.runtime.tree.Tree;
import tom.library.utils.Viewer;
import tom.library.sl.*;
import java.util.*;
import java.lang.*;
import java.io.*;

public class Main {
    %include{sl.tom}
    %include{util/HashMap.tom}
    %include{util/types/Collection.tom}
    %include{../genI/gram/i/i.tom}

    public static void main(String[] args) {
        try {
            iLexer lexer = new iLexer(new ANTLRInputStream(System.in));
            CommonTokenStream tokens = new CommonTokenStream(lexer);
            iParser parser = new iParser(tokens);
            // Parse the input expression
            Tree b = (Tree) parser.prog().getTree();
            //System.out.println("Result = " + iAdaptor.getTerm(b)); //
            // name of the Gom module + Adaptor
            Instrucao p = (Instrucao) iAdaptor.getTerm(b);

            Main main = new Main();
            String instrucoes = main.compileAnnot(p);

            /* Export this representation to .dot file*/
            /*
            try{
                FileWriter out=new FileWriter(args[1]);
                Viewer.toDot(p,out);
            }
            catch (IOException e){
                System.out.println("ERROR in dot file");
            }
            */
            /*Export code generated to .txt file*/
            if (args.length > 0) {
                try {
```



```
        PrintWriter pw = new PrintWriter(args[0]);
        pw.print(instrucoes);
        pw.flush(); pw.close();
    }
    catch (IOException e){
        System.err.println("exception:␣" + e);
        return;
    }
}
else {
    System.out.println(instrucoes);
}
} catch (Exception e) {
    e.printStackTrace();
}
}

private String compileAnnot(Instrucao inst) {
    NumToInt numInstrucao = new NumToInt(1);
    String toReturn = compileAnnotInstrucao(inst, numInstrucao);
    return toReturn.concat(" Halt");
}

private String compileAnnotInstrucao(Instrucao i, NumToInt
numInstrucao) {
    %match(i) {
        Atribuicao(_, id, _, opAtrib, _, exp, _) -> {
            String genExp = 'compileAnnotExpressoes(exp, numInstrucao);

            %match(opAtrib) {
                Atrib() -> { return "Pushv␣\" + 'id + \"\",\" + genExp + \"
                    Store,\"+genAnnotation(numInstrucao.inc()); }
                Mult() -> { return "Pushv␣\" + 'id + \"\",Pushv␣\" + 'id +
                    \"\",Load,\" + genExp + \"Mul,Store,\"+genAnnotation(
                    numInstrucao.inc()); }
                Div() -> { return "Pushv␣\" + 'id + \"\",Pushv␣\" + 'id +
                    \"\",Load,\" + genExp + \"Div,Store,\"+genAnnotation(
                    numInstrucao.inc()); }
                Soma() -> { return "Pushv␣\" + 'id + \"\",Pushv␣\" + 'id +
                    \"\",Load,\" + genExp + \"Add,Store,\"+genAnnotation(
                    numInstrucao.inc()); }
                Sub() -> { return "Pushv␣\" + 'id + \"\",Pushv␣\" + 'id +
                    \"\",Load,\" + genExp + \"Sub,Store,\"+genAnnotation(
                    numInstrucao.inc()); }
            }
            return "";
        }
    }

    Declaracao(_, tipo, _, decls, _, _) -> {
```



```
    return 'compileAnnotDeclaracoes(decls , tipo , numInstrucao);
}

If( , , , condicao , , , inst1 , inst2) -> {
    String genCondicao = 'compileAnnotExpressoes(condicao ,
        numInstrucao);
    String genInst1 = 'compileAnnotInstrucao(inst1 , numInstrucao)
        ;
    String genInst2 = 'compileAnnotInstrucao(inst2 , numInstrucao)
        ;

    return genCondicao + "Jumpf_\\"senao" + numInstrucao.get() + "
        \\", \" + genInst1 + "Jump_\\"fse" + numInstrucao.get() + "\\",
        ALabel_\\"senao" + numInstrucao.get() + "\\", \" + genInst2 +
        "ALabel_\\"fse" + numInstrucao.get() + "\\", \" + genAnnotation(
        numInstrucao.inc());
}

While( , , , condicao , , , inst , ) -> {
    String genCondicao = 'compileAnnotExpressoes(condicao ,
        numInstrucao);
    String genInst = 'compileAnnotInstrucao(inst , numInstrucao);

    return genCondicao + "Jumpf_\\"fenq" + numInstrucao.get() + "
        \\", \" + genInst + "ALabel_\\"fenq" + numInstrucao.get() + "
        \\", \" + genAnnotation(numInstrucao.inc());
}

For( , , , decl , , , condicao , , , exp , , , inst , ) -> {
    String genDecl = 'compileAnnotInstrucao(decl , numInstrucao);
    String genCondicao = 'compileAnnotExpressoes(condicao ,
        numInstrucao);
    String genExp = 'compileAnnotExpressoes(exp , numInstrucao);
    String genInst = 'compileAnnotInstrucao(inst , numInstrucao);

    return genDecl.concat(genCondicao).concat(genExp).concat(
        genInst);
}

Return( , , , exp , ) -> {
    return 'compileAnnotExpressoes(exp , numInstrucao);
}

Print( , , , exp , , ) -> {
    String genExp = 'compileAnnotExpressoes(exp , numInstrucao);
    return genExp + "IOut,\";
}

Funcao( , , tipo , , , nome , , , argumentos , , , inst , ) -> {
```




```
        return 'compileAnnotInstrucao(inst , numInstrucao);
    }

    Exp(exp) -> {
        return 'compileAnnotExpressoes(exp , numInstrucao);
    }

    SeqInstrucao(inst1 , inst*) -> {
        String genInst = 'compileAnnotInstrucao(inst1 , numInstrucao);

        return genInst.concat('compileAnnotInstrucao(inst*,
                                numInstrucao));
    }
}
return "";
}

private String compileAnnotDeclaracoes(Declaracoes decl , DefTipo
    tipo , NumToInt numInstrucao) {
    %match(decl) {
        ListaDecl(decl , tail*) -> {
            String gen = 'compileAnnotDeclaracoes(decl , tipo ,
                numInstrucao);
            String gen2 = 'compileAnnotDeclaracoes(tail* , tipo ,
                numInstrucao);
            gen.concat(gen2);

            return gen;
        }
    }

    Decl(id , _ , _ , exp , _) -> {
        String genExp = 'compileAnnotExpressoes(exp , numInstrucao);

        %match(tipo) {
            DInt() -> { return "Decl_\\" + 'id + "\\"_"+genExp; }
            DChar() -> { return "Decl_\\" + 'id + "\\"_"+genExp; }
            DBoolean() -> { return "Decl_\\" + 'id + "\\"_"+genExp; }
            DFloat() -> { return "Decl_\\" + 'id + "\\"_"+genExp; }
            DVoid() -> { return "Decl_\\" + 'id + "\\"_"+genExp; }
        }
        return "";
    }
}

return "";
}

private String compileAnnotExpressoes(Expressao e , NumToInt
    numInstrucao) {
    %match(e) {
        ExpNum(exp1 , _ , op , _ , exp2) -> {
```



```
String genExp1 = 'compileAnnotExpressoes(exp1, numInstrucao);
String genExp2 = 'compileAnnotExpressoes(exp2, numInstrucao);

%match(op) {
    Mais() -> { return genExp1 + genExp2 + "Add,"+genAnnotation
                (numInstrucao.inc()); }
    Vezes() -> { return genExp1 + genExp2 + "Mul,"+
                genAnnotation(numInstrucao.inc()); }
    Divide() -> { return genExp1 + genExp2 + "Div,"+
                genAnnotation(numInstrucao.inc()); }
    Menos() -> { return genExp1 + genExp2 + "Sub,"+
                genAnnotation(numInstrucao.inc()); }
    Mod() -> { return genExp1.concat(genExp2); }
}
return "";
}

Id(id) -> { return "Pushv_\\"" + 'id + "\",Load,"; }

Pos(exp) -> { return 'compileAnnotExpressoes(exp, numInstrucao)
; }

Neg(exp) -> { return 'compileAnnotExpressoes(exp, numInstrucao)
; }

Nao(exp) -> {
    String genExp = 'compileAnnotExpressoes(exp, numInstrucao);
    return genExp + "Not";
}

Call(_,id,_,_,parametros,_,_) -> { return ""; }

IncAntes(opInc,id) -> {
    %match(opInc) {
        Inc() -> { return "Pushv_\\"" + 'id + "\",Inc,Pushv_\\"" + '
                    id + "\",Load,"; }
        Dec() -> { return "Pushv_\\"" + 'id + "\",Dec,Pushv_\\"" + '
                    id + "\",Load,"; }
    }
    return 'id;
}

IncDepois(opInc,id) -> {
    %match(opInc) {
        Inc() -> { return "Pushv_\\"" + 'id + "\",Load,Pushv_\\"" + '
                    id + "\",Inc,"; }
        Dec() -> { return "Pushv_\\"" + 'id + "\",Load,Pushv_\\"" + '
                    id + "\",Dec,"; }
    }
}
```



```
    return 'id';
}

Condicional(condicao,_,_,exp1,_,_,exp2) -> {
    String genCondicao = 'compileAnnotExpressoes(condicao,
        numInstrucao);
    String genExp1 = 'compileAnnotExpressoes(exp1, numInstrucao);
    String genExp2 = 'compileAnnotExpressoes(exp2, numInstrucao);

    return genCondicao.concat(genExp1).concat(genExp2);
}

Int(i) -> { return "Pushi_" + 'i + ","; }

Char(c) -> { return "Pushc_" + 'c.charAt(0) + "'',"; }

True() -> { return "Pushb_true,"; }

False() -> { return "Pushb_false,"; }

Float(f) -> { return "Pushf_" + 'f + ","; }

Ou(cond1,_,_,cond2) -> {
    String genCond1 = 'compileAnnotExpressoes(cond1, numInstrucao
    );
    String genCond2 = 'compileAnnotExpressoes(cond2, numInstrucao
    );

    return genCond1 + genCond2 + "Or,"+genAnnotation(numInstrucao
        .inc());
}

E(cond1,_,_,cond2) -> {
    String genCond1 = 'compileAnnotExpressoes(cond1, numInstrucao
    );
    String genCond2 = 'compileAnnotExpressoes(cond2, numInstrucao
    );

    return genCond1 + genCond2 + "And,"+genAnnotation(
        numInstrucao.inc());
}

Comp(exp1,_,_,opComp,_,_,exp2) -> {
    String genExp1 = 'compileAnnotExpressoes(exp1, numInstrucao);
    String genExp2 = 'compileAnnotExpressoes(exp2, numInstrucao);

    %match(opComp) {
        Maior() -> { return genExp1 + genExp2 + "Gt,"+genAnnotation
            (numInstrucao.inc()); }
    }
```



```
Menor() -> { return genExp1 + genExp2 + "Lt,"+genAnnotation
            (numInstrucao.inc()); }
MaiorQ() -> { return genExp1 + genExp2 + "GoEq,"+
            genAnnotation(numInstrucao.inc()); }
MenorQ() -> { return genExp1 + genExp2 + "LoEq,"+
            genAnnotation(numInstrucao.inc()); }
Dif() -> { return genExp1 + genExp2 + "Neq,"+genAnnotation(
            numInstrucao.inc()); }
Igual() -> { return genExp1 + genExp2 + "Eq,"+genAnnotation
            (numInstrucao.inc()); }
    }
}

Input(,,_,tipo,_,_) -> {
    %match(tipo) {
        DInt() -> { return "IIn_int,"; }
        DChar() -> { return "IIn_char,"; }
        DBoolean() -> { return "IIn_boolean,"; }
        DFloat() -> { return "IIn_float,"; }
    }
}

Empty() -> { return ","; }
}
return "";
}

private String genAnnotation(int i) {
    if (i == 1) {
        return "Pushi_"+i+",IOut,";
    }
    else {
        return "Pushc_'',IOut,Pushi_"+i+",IOut,";
    }
}
}

class NumToInt{
    private int num;

    public NumToInt(int num) {
        this.num = num;
    }

    public NumToInt(){
        num = 0;
    }

    public int inc(){
```



```
    return num++;  
}  
  
public int get() {  
    return num;  
}  
}
```

F Gramática para ANTLR Máquina Virtual com construtores Gom-Tom



```
|      'IIn' idTipo -> ^(IIn idTipo)                //IO
|      'IOut' -> ^(IOut)
//
|      'Jump' STRING -> ^(Jump STRING)              //Jump
Instructions
|      'Jumpf' STRING -> ^(Jumpf STRING)
//
|      'Pushv' STRING -> ^(Push ^(S STRING))        //
Stack Operations
|      'Pushi' INT -> ^(Push ^(I INT))
|      'Pushc' CHAR -> ^(Push ^(S CHAR))
|      'Pushf' FLOAT -> ^(Push ^(F FLOAT))
|      'Pushb' boolean_ -> ^(Push boolean_)
|      'Load' -> ^(Load)
|      'Store' -> ^(Store)
|      'Decl' STRING valor -> ^(Decl STRING valor )
;

valor
:      FLOAT -> ^(F FLOAT)
|      INT -> ^(I INT)
|      STRING -> ^(S STRING)
|      boolean_ -> ^(B boolean_)
|      -> ^(Vazio)
;

boolean_
:      'true' -> ^(True)
|      'false' -> ^(False)
;

idTipo :      ('char' -> ^(DChar) | 'int' -> ^(DInt) | 'boolean' ->
              ^ (DBoolean) | 'float' -> ^(DFloat) )
;

// Tokens lex

STRING
@after
{
    setText(getText().substring(1, getText().length()-1));
}
:      '\\"' ~('\"')* '\\"'
;

CHAR
@after
{
    setText(getText().substring(1, getText().length()-1));
```



```
}
    :      '\'' ( '\'' ('b'|'t'|'n'|'f'|'r'|'\''|'\''|'\''|'\'' ) |
      ~('\''|'\'' ) ) '\'',
    ;

fragment DIGITO
    :      ('0'..'9')+
    ;

FLOAT
    :      DIGITO+ '.' DIGITO* SuffixoFloat?
    |      '.' DIGITO+ SuffixoFloat?
    |      INT SuffixoFloat
    ;

SuffixoFloat
    :      'f'|'F'
    ;

INT
    :      ('0' | '1'..'9' DIGITO*)
    ;

ID
    :      LETRA ( LETRA | '0'..'9' )*
    ;

fragment LETRA
    :      'a'..'z' | 'A'..'Z' | '_'
    ;

WS :      (' '|'\r'|'\t'|'\u000C'|'\n') {$channel=HIDDEN;}
    ;
```




G Código Gom Máquina Virtual

```
module maqv.msp
imports int String
abstract syntax

Instrucoes = Instrucoes(Instrucao*)

Instrucao = ALabel(id:String)
          | Call(id:String)
          | Ret()
          | Add()
          | Sub()
          | Div()
          | Mul()
          | Mod()
          | Inc()
          | Dec()
          | Eq()
          | Neq()
          | Gt()
          | GoEq()
          | Lt()
          | LoEq()
          | Nott()
          | Or()
          | And()
          | Halt()
          | IIn(tipo:DefTipo)
          | IOut()
          | Jump(id:String)
          | Jumpf(id:String)
          | Push(t:Termo)
          | Load()
          | Store()
          | Decl(id:String, valor:Termo)

DefTipo = DInt() | DChar() | DBoolean() | DFloat()

Boool = True() | False()

Stackk = Stackk(Termo*)

Termo = I(i:int)
      | S(id:String)
      | B(b:Boool)
      | F(f:int)
      | Vazio()
```



H Código Main Máquina Virtual

```
package maqv;

import maqv.msp.mspAdaptor;
import maqv.msp.types.*;
import org.antlr.runtime.CommonTokenStream;
import org.antlr.runtime.ANTLRInputStream;
import org.antlr.runtime.tree.Tree;
import tom.library.utils.Viewer;
import tom.library.sl.*;
import java.util.*;
import java.lang.*;
import java.io.*;

public class Main {
    %include{sl.tom}
    %include{util/HashMap.tom}
    %include{util/types/Collection.tom}
    %include{../genMaqV/maqv/msp/msp.tom}

    private Instrucoes programa;
    private Stackk stack;
    private Map<String,Termo> heap;
    private int pc;
    private int numProg;
    private StringBuilder output;

    public static void main(String[] args) {
        try {
            mspLexer lexer = new mspLexer(new ANTLRInputStream(new
                FileInputStream(args[0])));
            CommonTokenStream tokens = new CommonTokenStream(lexer);
            mspParser parser = new mspParser(tokens);
            // Parse the input expression
            Tree b = (Tree) parser.programa().getTree();
            //System.out.println("Result = " + mspAdaptor.getTerm(b)); //
                name of the Gom module + Adaptor
            Instrucoes p = (Instrucoes) mspAdaptor.getTerm(b);

            Main main = new Main(p);

            main.run(p);

            /* Export this representation to .dot file*/
            /*
            try{
                FileWriter out=new FileWriter("gram.dot");
```



```
        Viewer.toDot(p, out);
    }
    catch (IOException e){
        System.out.println("ERROR in dot file");
    }
    */
    if (args.length > 1) {
        try {
            PrintWriter pw = new PrintWriter(new BufferedWriter(new
                FileWriter(args[1], true)));
            pw.print(main.getOutput());
            pw.flush(); pw.close();
        }
        catch (IOException e){
            System.err.println("exception:_" + e);
            return;
        }
    }
    else {
        System.out.println(main.getOutput());
    }
} catch (Exception e) {
    e.printStackTrace();
}
}

public Main(Instrucoes insts) {
    programa = insts;
    stack = 'Stackk();
    heap = new HashMap<String , Termo>();
    pc = 0;
    numProg = 0;
    output = new StringBuilder();
}

public String getOutput(){
    return output.toString();
}

private Instrucoes getNInstr(Instrucoes prog, int pc){
    %match (prog){
        i@Instrucoes(inst , insts*) -> {
            if (pc == 0) {
                return 'i;
            }
            else { return 'getNInstr(insts*,pc-1); }
        }
    }
    return 'Instrucoes();
}
```



```
}

private Instrucoes jmp(Instrucoes prog, String label){
    %match (prog){
        Instrucoes(ALabel(l), insts*) -> {
            if (label.equals('l')) { return 'insts*'; }
            else { return 'jmp(insts*,label); }
        }
        Instrucoes(_, insts*) -> { return 'jmp(insts*,label); }
    }
    return 'Instrucoes()';
}

private void pushStack(Termo termo){
    %match (stack){
        Stackk(terms*) -> { this.stack = 'Stackk(termo,terms*); }
    }
}

private void popStack(){
    %match (stack){
        Stackk(termo1, terms*) -> { this.stack = 'Stackk(terms*); }
    }
}

private Termo topStack() {
    %match(stack){
        Stackk(termo, terms*) -> { return 'termo'; }
    }
    return 'Vazio()';
}

public String run(Instrucoes prog) {
    %match (prog){
        Instrucoes(inst, instrs*) -> {
            %match(inst) {
                ALabel(id) -> { return 'run(instrs*); }
                Call(id) -> {
                    'pushStack(I(pc));
                    prog = 'jmp(prog, id);
                    return 'run(prog);
                }
            }
            Ret() -> {
                Termo progCount = 'topStack();
                %match(progCount) {
                    I(valor) -> { prog = 'getNInstr(prog, valor); }
                }
                return 'run(prog);
            }
        }
    }
}
```



```
Add() -> {
    %match (stack){
        Stackk(I(v2),I(v1),resto*) -> {
            stack = 'resto*';
            int resultado = 'v1+'v2;
            'pushStack(I(resultado));
            return 'run(instrs*);
        }
    }
}

Sub() -> {
    %match (stack){
        Stackk(I(v2),I(v1),resto*) -> {
            stack = 'resto*';
            int resultado = 'v1 - 'v2;
            'pushStack(I(resultado));
            return 'run(instrs*);
        }
    }
}

Div() -> {
    %match (stack){
        Stackk(I(v2),I(v1),resto*) -> {
            stack = 'resto*';
            int resultado = 'v1 / 'v2;
            'pushStack(I(resultado));
            return 'run(instrs*);
        }
    }
}

Mul() -> {
    %match (stack){
        Stackk(I(v2),I(v1),resto*) -> {
            stack = 'resto*';
            int resultado = 'v1 * 'v2;
            'pushStack(I(resultado));
            return 'run(instrs*);
        }
    }
}

Mod() -> {
    %match (stack){
        Stackk(I(v2),I(v1),resto*) -> {
            stack = 'resto*';
            int resultado = 'v1 % 'v2;
            'pushStack(I(resultado));
            return 'run(instrs*);
        }
    }
}
```



```
}
Inc() -> {
    Termo t = 'topStack();
    'popStack();
    %match(t) {
        S(id) -> {
            Termo t2 = heap.get('id);
            int valorToInc = 0;
            %match(t2) {
                I(v1) -> { valorToInc = 'v1+1; }
            }
            heap.put('id, 'I(valorToInc));

            return 'run(instrs*);
        }
    }
    return 'run(instrs*);
}
Dec() -> {
    Termo t = 'topStack();
    'popStack();
    %match(t) {
        S(id) -> {
            Termo t2 = heap.get('id);
            int valorToDec = 0;
            %match(t2) {
                I(v1) -> { valorToDec = 'v1-1; }
            }
            heap.put('id, 'I(valorToDec));

            return 'run(instrs*);
        }
    }
    return 'run(instrs*);
}
Eq() -> {
    %match (stack){
        Stackk(I(v2), I(v1), resto*) -> {
            stack = 'resto*;
            Bool resultado = ('v1 == 'v2) ? 'True() : 'False();
            'pushStack(B(resultado));
            return 'run(instrs*);
        }
    }
}
Neq() -> {
    %match (stack){
        Stackk(I(v2), I(v1), resto*) -> {
            stack = 'resto*;
```



```
        Bool resultado = ('v1 == 'v2) ? 'False() : 'True();
        'pushStack(B(resultado));
        return 'run(instrs*);
    }
}
}
Gt() -> {
    %match (stack){
        Stackk(I(v2),I(v1),resto*) -> {
            stack = 'resto*;
            Bool resultado = ('v1 > 'v2) ? 'True() : 'False();
            'pushStack(B(resultado));
            return 'run(instrs*);
        }
    }
}
GoEq() -> {
    %match (stack){
        Stackk(I(v2),I(v1),resto*) -> {
            stack = 'resto*;
            Bool resultado = ('v1 >= 'v2) ? 'True() : 'False();
            'pushStack(B(resultado));
            return 'run(instrs*);
        }
    }
}
Lt() -> {
    %match (stack){
        Stackk(I(v2),I(v1),resto*) -> {
            stack = 'resto*;
            Bool resultado = ('v1 < 'v2) ? 'True() : 'False();
            'pushStack(B(resultado));
            return 'run(instrs*);
        }
    }
}
LoEq() -> {
    %match (stack){
        Stackk(I(v2),I(v1),resto*) -> {
            stack = 'resto*;
            Bool resultado = ('v1 <='v2) ? 'True() : 'False();
            'pushStack(B(resultado));
            return 'run(instrs*);
        }
    }
}
Nott() -> {
    Termo t = 'topStack();
    'popStack();
}
```



```
%match(t) {
    i@I(v1) -> { 'pushStack(i); }
    s@S(v1) -> { 'pushStack(s); }
    f@F(v1) -> { 'pushStack(f); }
    B(True()) -> { 'pushStack(B(False())); }
    B(False()) -> { 'pushStack(B(True())); }
}
return 'run(instrs*);
}
Or() -> {
    %match (stack){
        Stackk(B(v2),B(v1),resto*) -> {
            stack = 'resto*';
            boolean valor1 = true, valor2 = true;
            %match (v1){
                True() -> { 'valor1 = true; }
                False() -> { 'valor1 = false; }
            }
            %match (v2){
                True() -> { 'valor2 = true; }
                False() -> { 'valor2 = false; }
            }
            Bool resultado = (valor1 || valor2) ? 'True()' : '
                False()';
            'pushStack(B(resultado));
            return 'run(instrs*);
        }
    }
}
And() -> {
    %match (stack){
        Stackk(B(v2),B(v1),resto*) -> {
            stack = 'resto*';
            boolean valor1 = true, valor2 = true;
            %match (v1){
                True() -> { 'valor1 = true; }
                False() -> { 'valor1 = false; }
            }
            %match (v2){
                True() -> { 'valor2 = true; }
                False() -> { 'valor2 = false; }
            }
            Bool resultado = (valor1 && valor2) ? 'True()' : '
                False()';
            'pushStack(B(resultado));
            return 'run(instrs*);
        }
    }
}
}
```




```
Halt() -> { return ""; }
IIn(tipo) -> {
    BufferedReader br = new BufferedReader(new
        InputStreamReader(System.in));
    try{
        String iin = br.readLine();
        %match(tipo) {
            DInt() -> {
                try{
                    int v1 = Integer.parseInt(iin);
                    'pushStack(I(v1));
                }catch(NumberFormatException nfe){
                    System.err.println("Invalid Format!");
                }
            }
            DChar() -> {
                String s = ""+iin.charAt(0);
                'pushStack(S(s));
            }
            DBoolean() -> {
                if (iin.equalsIgnoreCase("true")) { 'pushStack(B(
                    True())); }
                else if (iin.equalsIgnoreCase("false")) { '
                    pushStack(B(False())); }
                else { System.err.println("Invalid Format!"); }
            }
            DFloat() -> {
                try{
                    int v1 = Integer.parseInt(iin);
                    'pushStack(I(v1));
                }catch(NumberFormatException nfe){
                    System.err.println("Invalid Format!");
                }
            }
        }
    }
    catch (IOException e){
        System.err.println("exception:_" + e);
    }
    return 'run(instrs*);
}
IOut() -> {
    Termo t = 'topStack();
    'popStack();
    %match(t) {
        I(v1) -> { output.append('v1); }
        S(v1) -> { output.append('v1); }
        F(v1) -> { output.append('v1); }
        B(True()) -> { output.append("True"); }
    }
}
```



```
        B(False()) -> { output.append("False"); }
    }
    return 'run(instrs*);
}
Jump(id) -> {
    prog = 'jmp(prog, id);
    return 'run(prog);
}
Jumpf(id) -> {
    Termo t = 'topStack();
    'popStack();
    %match(t) {
        B(True()) -> { return 'run(instrs*); }
        B(False()) -> {
            prog = 'jmp(prog, id);
            return 'run(prog);
        }
    }
    return 'run(instrs*);
}
Push(t) -> {
    'pushStack(t);
    return 'run(instrs*);
}
Load() -> {
    Termo t = 'topStack();
    'popStack();
    %match(t) {
        S(id) -> {
            Termo t2 = heap.get('id');
            'pushStack(t2);

            return 'run(instrs*);
        }
    }
    return 'run(instrs*);
}
Store() -> {
    Termo t = 'topStack();
    'popStack();
    Termo t2 = 'topStack();
    'popStack();
    %match(t2) {
        S(id) -> {
            heap.put('id', t);

            return 'run(instrs*);
        }
    }
}
```



```
        return 'run(instrs*);
    }
    Decl(id, valor) -> {
        heap.put('id', 'valor');
        return 'run(instrs*);
    }
}
}
}
return "";
}
```



I Código java da aplicação de detecção de falhas

```
package ats;

import com.itextpdf.text.Anchor;
import com.itextpdf.text.BadElementException;
import com.itextpdf.text.BaseColor;
import com.itextpdf.text.Chapter;
import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import com.itextpdf.text.Element;
import com.itextpdf.text.Font;
import com.itextpdf.text.Paragraph;
import com.itextpdf.text.Phrase;
import com.itextpdf.text.Section;
import com.itextpdf.text.pdf.PdfPCell;
import com.itextpdf.text.pdf.PdfPTable;
import com.itextpdf.text.pdf.PdfWriter;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.StringTokenizer;
import java.util.TreeSet;

/**
 *
 * @author pg22785 ; pg22827; pg23093
 */

public class ATS {

    private Integer max =0;
    private ArrayList <TreeSet<Integer>> dados = new ArrayList <>();
    private ArrayList <String> resultados = new ArrayList <>();
    private ArrayList <String> resultadosEsperados = new ArrayList
        <>();

    public ATS() {
    }

    public Integer getMax() {
        return max;
    }
}
```



```
public boolean lerFecheiroTeste(String file) throws
FileNotFoundException, IOException{
    this.dados.clear();
    this.resltados.clear();
    this.max = 0;
    boolean t = true;
    BufferedReader br = new BufferedReader(new FileReader(file));
    while(br.ready() && t){
        String linha = br.readLine();
        t = partir(linha);
    }
    br.close();
    return t;
}

public void lerFecheiroRes(String file) throws
FileNotFoundException, IOException{
    this.resltadosEsperados.clear();
    BufferedReader br = new BufferedReader(new FileReader(file));
    while(br.ready()){
        String linha = br.readLine();
        this.resltadosEsperados.add(linha);
    }
    br.close();
}

public boolean tamanho(){
    boolean aux;
    if(dados.size() == resltadosEsperados.size()) {
        aux = true;
    }
    else {
        aux=false;
    }
    return aux;
}

public boolean partir(String linha){
    boolean t;
    TreeSet<Integer> aux = new TreeSet<>();
    StringTokenizer str = new StringTokenizer(linha, ",");
    String blocos = str.nextToken();
    if(str.hasMoreElements()){
        StringTokenizer str2 = new StringTokenizer(blocos, ",");
        Integer i = 0;
        t = true;
        while(str2.hasMoreElements() && t){
            try {
                i = Integer.parseInt(str2.nextToken());
            }
        }
    }
}
```



```
        } catch (NumberFormatException nfe) {
            t = false;
        }
        if (t) {
            if (i > this.max) {
                this.max = i;
            }
            aux.add(i);
        }
        this.resltados.add(str.nextToken());
        this.dados.add(aux);
    }
    else {
        t = false;
    }
    return t;
}

public ArrayList <ArrayList<Integer>> initMatris () {
    ArrayList <ArrayList<Integer>> tablea = new ArrayList <
        ArrayList<Integer>>();
    for (int i = 0; i < this.dados.size(); i++) {
        ArrayList<Integer> linhaTab = new ArrayList <>();
        for (int j = 0; j < this.max; j++) {
            linhaTab.add(0);
        }
        tablea.add(linhaTab);
    }
    return tablea;
}

public ArrayList <ArrayList<Integer>> matris () {
    ArrayList <ArrayList<Integer>> tablea = initMatris();
    Iterator it1 = dados.iterator();
    int i = 0;
    while (it1.hasNext())
    {
        TreeSet<Integer> aux = (TreeSet<Integer>) it1.next();
        Iterator it = aux.iterator();
        while (it.hasNext()) {
            tablea.get(i).set((Integer) it.next() - 1, 1);
        }
        i++;
    }
    return tablea;
}

public ArrayList<Integer> res () {
```



```
        ArrayList<Integer> aux = new ArrayList<>();
        Iterator it1 = resultados.iterator();
        Iterator it2 = resultadosEsperados.iterator();
        while(it1.hasNext() && it2.hasNext()){
            if(it1.next().equals(it2.next())) {
                aux.add(0);
            }
            else {
                aux.add(1);
            }
        }
        return aux;
    }

    public ArrayList<Float> probabilidades (ArrayList <ArrayList<
        Integer>> matris , ArrayList<Integer> res){
        ArrayList<Float> prob = new ArrayList<>();
        float aux;
        int n11 = 0;
        int n10 = 0;
        int n01 = 0;
        for (int i = 0; i < matris.get(0).size() ; i++ ){
            for(int j = 0; j < res.size() ;j++){
                int a = matris.get(j).get(i);
                int b = res.get(j);
                if( a == 1 && b == 1 ) {
                    n11++;
                }
                else if ( a == 1 && b == 0 ) {
                    n10++;
                }
                else if ( a == 0 && b == 1 ) {
                    n01++;
                }
            }
            aux = (n11 + n10 + n01);
            aux = (n11 /aux);
            prob.add(aux);
            n11 = 0;
            n10 = 0;
            n01 = 0;
        }
        return prob;
    }

    private void show(ArrayList <ArrayList<Integer>> matris , ArrayList
        <Integer> res){
        Iterator it1 = matris.iterator();
        Iterator it2 = res.iterator();
```



```
        while (it1.hasNext() && it2.hasNext())
        {
            ArrayList<Integer> aux = (ArrayList<Integer>) it1.next();
            Iterator it3 = aux.iterator();
            while (it3.hasNext()){
                System.out.print(it3.next());
                System.out.print(" |");
            }
            System.out.print(it2.next());
            System.out.println();
        }
        System.out.println(probabilidades(matris, res));
    }

    //      public static void main(String[] args) throws
    //      FileNotFoundException, IOException {
    //          ATS teste = new ATS();
    //          teste.lerFecheiroTeste("testes.txt");
    //          teste.lerFecheiroRes("res_esp.txt");
    //          teste.show(teste.matris(), teste.res());
    //      }

    public void gerarRelatorio(String caminho) throws
    FileNotFoundException, DocumentException{
        String FILE = caminho + "/FaultLocalization.pdf";
        Font catFont = new Font(Font.FontFamily.TIMES_ROMAN, 18, Font
        .BOLD);
        Font subFont = new Font(Font.FontFamily.TIMES_ROMAN, 16, Font
        .BOLD);
        Font smallBold = new Font(Font.FontFamily.TIMES_ROMAN, 12,
        Font.BOLD);
        try {
            Document doc = new Document();
            PdfWriter.getInstance(doc, new FileOutputStream(FILE));
            doc.open();
            addMetaData(doc);
            addTitlePage(doc, catFont, smallBold);
            addContent(doc, catFont, subFont);
            doc.close();
        } catch (FileNotFoundException | DocumentException e) {}
    }

    private void addMetaData(Document document) {
        document.addTitle("Spectrum-based_Fault_Localization");
        document.addSubject("Matrizes");
        document.addAuthor("Bruno, Luis e Tiago");
        document.addCreator("Bruno, Luis e Tiago");
    }
}
```




```
private void addTitlePage(Document document, Font catFont, Font
    smallBold) throws DocumentException{
    Paragraph preface = new Paragraph();
    addEmptyLine(preface, 1);
    preface.add(new Paragraph("Spectrum-based Fault Localization"
        , catFont));
    addEmptyLine(preface, 4);
    preface.add(new Paragraph("Este documento demonstra a matriz
        criada pela aplicacao assim como o vector de erros.",
        smallBold));
    addEmptyLine(preface, 8);
    document.add(preface);
}

private void addContent(Document document, Font catFont, Font
    subFont) throws DocumentException {
    Anchor anchor = new Anchor("Matriz", catFont);
    anchor.setName("Matriz");
    // Second parameter is the number of the chapter
    Chapter catPart = new Chapter(new Paragraph(anchor), 1);
    // Add a table
    createTable(catPart);
    // Now add all this to the document
    document.add(catPart);
}

private void createTable(Chapter CatPart) throws
    BadElementException {
    int colunas = this.getMax() + 2; //2 corresponde ao teste e ao
        erro
    int cont = 0, nr_tab = colunas/7, resto = colunas%7, i = 0;

    ArrayList<ArrayList<Integer>> matriz = this.matris();
    ArrayList<Integer> res = this.res();
    ArrayList<Float> probab = this.probabilidades(matriz, res);
    for (; cont<nr_tab; cont++){ //vai fazer "nr_tab" tabelas
        if(cont == 0){ //Na primeira tabela tem que meter a
            coluna teste
            int lim = 6*(cont+1);
            PdfPTable table = new PdfPTable(7);
            //colunas
            table.addCell("Teste");
            for (; i < lim; i++){
                table.addCell("B" + (i+1));
            }
            //linhas
            int j = 1;
            Iterator it = matriz.iterator();
            while(it.hasNext()){
```



```
        table.addCell(""+j);
        ArrayList<Integer> aux = (ArrayList<Integer>) it.
            next();
        for(i = 0; i < lim; i++){
            table.addCell(""+ aux.get(i));
        }
        j++;
    }
    //adicionar probabilidades
    table.addCell("PE");
    for(i = 0; i < lim; i++){
        table.addCell(""+ prob.get(i));
    }
    //adicionar tabela
    CatPart.add(table);
    CatPart.newPage();
}
else{
    if(resto != 0){//Garantir que o erro nao aparece aqui
        int aux_nr = i; int lim = 7*(cont+1)-1;
        PdfPTable table = new PdfPTable(7);
        //colunas
        for(; i < lim; i++){
            table.addCell("B" + (i+1));
        }
        //linhas
        for(ArrayList<Integer> aux: matriz){
            for(i = aux_nr; i < lim; i++){
                table.addCell(""+ aux.get(i));
            }
        }
        //adicionar probabilidades
        for(i = aux_nr; i < lim; i++){
            table.addCell(""+ prob.get(i));
        }
        //adicionar tabela
        CatPart.add(table);
        CatPart.newPage();
    }
    else{//Erro na ultima coluna; implica que resto e' 0
        if((cont+1) == nr_tab){
            int aux_nr = i; int lim = 7*(cont+1)-2;
            PdfPTable table = new PdfPTable(7);
            //colunas
            for(; i < lim; i++){
                table.addCell("B" + (i+1));
            }
            table.addCell("Erro");
            //linhas
```



```
        int linh = 0;
        for(ArrayList<Integer> aux: matriz){
            for(i = aux_nr; i < lim; i++){
                table.addCell("" + aux.get(i));
            }
            //adicionar erro
            table.addCell("" + res.get(linh));
            linh++;
        }
        //adicionar probabilidades
        for(i = aux_nr; i < lim; i++){
            table.addCell("" + prob.get(i));
        }
        table.addCell("");
        //adicionar tabela
        CatPart.add(table);
        CatPart.newPage();
    }
    else{ //Ainda nao e' a ultima tabela
        int aux_nr = i; int lim = 7*(cont+1)-1;
        PdfPTable table = new PdfPTable(7);
        //colunas
        for(; i < lim; i++){
            table.addCell("B" + (i+1));
        }
        //linhas
        for(ArrayList<Integer> aux: matriz){
            for(i = aux_nr; i < lim; i++){
                table.addCell("" + aux.get(i));
            }
        }
        //adicionar probabilidades
        for(i = aux_nr; i < lim; i++){
            table.addCell("" + prob.get(i));
        }
        //adicionar tabela
        CatPart.add(table);
        CatPart.newPage();
    }
}
}
}
}
if(resto>0){ //Se ainda sobrar colunas mas que nao chegam a 7
    colunas
    int aux_nr = i;
    PdfPTable table = new PdfPTable(resto);
    //colunas
    for(; i < (colunas-2); i++){
        table.addCell("B" + (i+1));
    }
}
```



```
    }
    table.addCell("Erro");
    //linhas
    int linh = 0;
    for( ArrayList<Integer> aux: matriz){
        for(i = aux_nr; i < (colunas-2); i++){
            table.addCell(" " + aux.get(i));
        }
        //adicionar erro
        table.addCell(" " + res.get(linh));
        linh++;
    }
    //adicionar probabilidades
    for(i = aux_nr; i < (colunas-2); i++){
        table.addCell(" " + prob.get(i));
    }
    table.addCell(" ");
    //adicionar tabela
    CatPart.add(table);
    /*PdfPTable table = new PdfPTable(resto);
    int lim = res.size(), aux_nr = i;
    for(int z = 0; z < resto-1; z++){
        table.addCell("B" + (i+1));
        //linhas
        for( ArrayList<Integer> aux: matriz){
            for(i = aux_nr; i < (colunas-2); i++){
                table.addCell(" " + aux.get(i));
            }
        }
        //adicionar probabilidades
        for(i = aux_nr; i < (colunas-2); i++){
            table.addCell(" " + prob.get(i));
        }
    }
    //Ultima coluna de erros
    table.addCell("Erro");
    i = 0;
    while(i<lim){
        table.addCell(" " + res.get(i));
        i++;
    }
    CatPart.add(table);*/
}
}

private static void addEmptyLine(Paragraph paragraph, int number)
{
    for (int i = 0; i < number; i++) {
        paragraph.add(new Paragraph(" "));
    }
}
```



}
}
}



J Código Main i- v3

```
package gram;

import gram.i.iAdaptor;
import gram.i.types.*;
import org.antlr.runtime.CommonTokenStream;
import org.antlr.runtime.ANTLRInputStream;
import org.antlr.runtime.tree.Tree;
import tom.library.utils.Viewer;
import tom.library.sl.*;
import java.util.*;
import java.lang.*;
import java.io.*;

public class Main {
    %include{sl.tom}
    %include{util/HashMap.tom}
    %include{util/ArrayList.tom}
    %include{util/types/Collection.tom}
    %include{util/types/Set.tom}
    %include{../genI/gram/i/i.tom}

    private String actualFunctionName;
    HashMap<String, Argumentos> functionSignatures;
    private boolean callReturnNeeded;
    private int memAdress;
    StringBuilder functionsDeclarations;

    public static void main(String[] args) {
        try {
            iLexer lexer = new iLexer(new ANTLRInputStream(System.in));
            CommonTokenStream tokens = new CommonTokenStream(lexer);
            iParser parser = new iParser(tokens);
            // Parse the input expression
            Tree b = (Tree) parser.prog().getTree();
            //System.out.println("Result = " + iAdaptor.getTerm(b)); //
            // name of the Gom module + Adaptor
            Instrucao p = (Instrucao) iAdaptor.getTerm(b);

            Main main = new Main();

            try {
                ArrayList<Integer> numInstrucao = new ArrayList<Integer>();
                numInstrucao.add(1);
                // Instrucao p1 = 'TopDown(stratFaultInjection()).visit(p);
                'TopDown(CollectFuncsSignature(main.functionSignatures)).
                    visit(p);
            }
        }
    }
}
```



```
Instrucao p2 = 'BottomUp(stratPrintAnnotations(numInstrucao))
    .visit(p);
int numInst = numInstrucao.get(0)-1;
LComentarios c = 'Vazio();
Expressao numInstExps = 'Expressoes(Print(c,c,c,Int(numInst),
    c,c),Print(c,c,c,Char("#"),c,c));
NumToInt n = new NumToInt(1);
String numInstString = main.compileAnnotExpressoes(
    numInstExps, n);
String instrucoes = "";
if (args.length > 0) {
    if (args[0].equals("-fi") && args.length > 1) {
        TreeSet<Integer> blocosMaisUsados = new TreeSet<Integer>
            >();

        if(Main.parseFile(args[1], blocosMaisUsados)) {
            numInstrucao.clear();
            numInstrucao.add(1);
            Instrucao p3 = 'BottomUp(
                stratFaultInjectionWithKnowledge(numInstrucao,
                blocosMaisUsados)).visit(p2);
            instrucoes = main.compileAnnot(p3);
        } else {
            System.out.println("Failed_to_parse_blocks");
        }
    }
    else if (args[0].equals("-bs")) {
        Instrucao p3 = 'TopDown(stratBadSmells()).visit(p);
        instrucoes = main.compileAnnot(p3);
    }
    else {
        instrucoes = main.compileAnnot(p2);
    }
}
else {
    instrucoes = main.compileAnnot(p2);
}
String functionDeclarationsAndArguments = main.
    functionsDeclarations.toString();
System.out.println(functionDeclarationsAndArguments +
    numInstString + instrucoes);
} catch(VisitFailure e) {
    System.out.println("the_strategy_failed");
}

/* Export this representation to .dot file*/
/*
try{
    FileWriter out=new FileWriter(args[1]);
```



```
        Viewer.toDot(p, out);
    }
    catch (IOException e){
        System.out.println("ERROR in dot file");
    }
    */
    /*Export code generated to .txt file*/
} catch(Exception e) {
    e.printStackTrace();
}
}

public Main() {
    actualFunctionName = "";
    functionSignatures = new HashMap<String , Argumentos>();
    callReturnNeeded = true;
    functionsDeclarations = new StringBuilder();
    memAdress = 0;
}

public static Argumentos removeArgumentosNaoUtilizados(Argumentos
    args , TreeSet<String> idsUtilizados) {
    %match(args) {
        ListaArgumentos(arg1 , tailArg*) -> {
            %match(arg1) {
                a@Argumento( , , , idArg , ) -> {
                    if (idsUtilizados.contains('idArg))
                        return 'ListaArgumentos(a , removeArgumentosNaoUtilizados
                            (tailArg* , idsUtilizados));
                    else
                        return removeArgumentosNaoUtilizados('tailArg* ,
                            idsUtilizados);
                }
            }
        }
    }
    return args;
}

%strategy stratBadSmells() extends Identity() {
    visit Instrucao {
        If(c1 , c2 , c3 , Nao(condicao) , c4 , c5 , inst1 , inst2) -> {
            return 'If(c1 , c2 , c3 , condicao , c4 , c5 , inst2 , inst1);
        }
        Funcao(c1 , tipo , c2 , nome , c3 , c4 , argumentos , c5 , c6 , inst , c7) -> {
            TreeSet<String> idsUtilizados = new TreeSet<String>();
            'TopDown(stratCollectIds(idsUtilizados)).visit('inst);
            Argumentos args = removeArgumentosNaoUtilizados('argumentos
                , idsUtilizados);
        }
    }
}
```




```
        return 'Funcao(c1, tipo, c2, nome, c3, c4, args, c5, c6, inst, c7);
    }
}

%strategy stratCollectIds(Set idsUtilizados) extends Identity() {
    visit Instrucao {
        Atribuicao(_, id, _, opAtrib, _, exp, _) -> {
            idsUtilizados.add('id');
        }
    }
    visit Expressao {
        Id(id) -> {
            idsUtilizados.add('id');
        }
        IncAntes(opInc, id) -> {
            idsUtilizados.add('id');
        }
        IncDepois(opInc, id) -> {
            idsUtilizados.add('id');
        }
    }
}

%strategy CollectFuncsSignature(signatures:HashMap) extends
    Identity() {
    visit Instrucao {
        Funcao(_, tipo, _, nome, _, _, argumentos, _, _, inst, _) -> {
            signatures.put('nome', 'argumentos');
        }
    }
}

%strategy stratPrintAnnotations(ArrayList numInstrucao) extends
    Identity() {
    visit Instrucao {
        i@Atribuicao(_, _, _, _, _, _, _) -> {
            int num = (Integer) numInstrucao.remove((int) 0);
            LComentarios c = 'Vazio();
            numInstrucao.add(num+1);
            if (num > 1)
                return 'SeqInstrucao(i, Exp(Print(c, c, c, Char(", "), c, c)), Exp(
                    Print(c, c, c, Int(num), c, c)));
            else
                return 'SeqInstrucao(i, Exp(Print(c, c, c, Int(num), c, c)));
        }
        i@If(_, _, _, condicao, _, _, inst1, inst2) -> {
            int num = (Integer) numInstrucao.remove((int) 0);
            LComentarios c = 'Vazio();
```



```
numInstrucao.add(num+1);
if (num > 1)
    return 'SeqInstrucao(i,Exp(Print(c,c,c,Char(", "),c,c)),Exp(
        Print(c,c,c,Int(num),c,c)));
else
    return 'SeqInstrucao(i,Exp(Print(c,c,c,Int(num),c,c)));
}
i@While(_,_,_,condicao,_,_,inst,_) -> {
    int num = (Integer) numInstrucao.remove((int) 0);
    LComentarios c = 'Vazio();
    numInstrucao.add(num+1);
    if (num > 1)
        return 'SeqInstrucao(i,Exp(Print(c,c,c,Char(", "),c,c)),Exp(
            Print(c,c,c,Int(num),c,c)));
    else
        return 'SeqInstrucao(i,Exp(Print(c,c,c,Int(num),c,c)));
}
i@For(_,_,decl,_,_,condicao,_,_,exp,_,_,inst,_) -> {
    int num = (Integer) numInstrucao.remove((int) 0);
    LComentarios c = 'Vazio();
    numInstrucao.add(num+1);
    if (num > 1)
        return 'SeqInstrucao(i,Exp(Print(c,c,c,Char(", "),c,c)),Exp(
            Print(c,c,c,Int(num),c,c)));
    else
        return 'SeqInstrucao(i,Exp(Print(c,c,c,Int(num),c,c)));
}
i@Return(_,_,exp,_) -> {
    int num = (Integer) numInstrucao.remove((int) 0);
    LComentarios c = 'Vazio();
    numInstrucao.add(num+1);
    if (num > 1)
        return 'SeqInstrucao(i,Exp(Print(c,c,c,Char(", "),c,c)),Exp(
            Print(c,c,c,Int(num),c,c)));
    else
        return 'SeqInstrucao(i,Exp(Print(c,c,c,Int(num),c,c)));
}
}
}
visit Expressao {
e@ExpNum(exp1,_,_,op,_,_,exp2) -> {
    int num = (Integer) numInstrucao.remove((int) 0);
    LComentarios c = 'Vazio();
    numInstrucao.add(num+1);
    if (num > 1)
        return 'Expressoes(e,Print(c,c,c,Char(", "),c,c),Print(c,c,c,
            Int(num),c,c));
    else
        return 'Expressoes(e,Print(c,c,c,Int(num),c,c));
}
}
```



```
e@Ou(cond1,_,_,cond2) -> {
  int num = (Integer) numInstrucao.remove((int) 0);
  LComentarios c = 'Vazio();
  numInstrucao.add(num+1);
  if (num > 1)
    return 'Expressoes(e,Print(c,c,c,Char(", "),c,c),Print(c,c,c,
      ,Int(num),c,c));
  else
    return 'Expressoes(e,Print(c,c,c,Int(num),c,c));
}
e@E(cond1,_,_,cond2) -> {
  int num = (Integer) numInstrucao.remove((int) 0);
  LComentarios c = 'Vazio();
  numInstrucao.add(num+1);
  if (num > 1)
    return 'Expressoes(e,Print(c,c,c,Char(", "),c,c),Print(c,c,c,
      ,Int(num),c,c));
  else
    return 'Expressoes(e,Print(c,c,c,Int(num),c,c));
}
e@Comp(exp1,_,_,opComp,_,exp2) -> {
  int num = (Integer) numInstrucao.remove((int) 0);
  LComentarios c = 'Vazio();
  numInstrucao.add(num+1);
  if (num > 1)
    return 'Expressoes(e,Print(c,c,c,Char(", "),c,c),Print(c,c,c,
      ,Int(num),c,c));
  else
    return 'Expressoes(e,Print(c,c,c,Int(num),c,c));
}
}
}

%strategy stratFaultInjection() extends Identity() {
  visit Instrucao {
  If(c1,c2,c3,condicao,c4,c5,inst1,inst2) -> {
    return 'If(c1,c2,c3,condicao,c4,c5,inst2,inst1);
  }
  While(c1,c2,c3,condicao,c4,c5,inst,c6) -> {
    return 'While(c1,c2,c3,Nao(condicao),c4,c5,inst,c6);
  }
  For(c1,c2,decl,c3,condicao,c4,c5,exp,c6,c7,inst,c8) -> {
    return 'For(c1,c2,decl,c3,Nao(condicao),c4,c5,exp,c6,c7,inst,c8
      );
  }
}
}
```



```
%strategy stratFaultInjectionWithKnowledge(ArrayList numInstrucao
,Set blocos) extends Identity() {
visit Instrucao {
i@Atribuicao(_,_,_,_,_,_,_) -> {
    int num = (Integer) numInstrucao.remove((int) 0);
    numInstrucao.add(num+1);
    if (blocos.contains((Integer) num))
        return 'i;
}
If(c1,c2,c3,condicao,c4,c5,inst1,inst2) -> {
    int num = (Integer) numInstrucao.remove((int) 0);
    numInstrucao.add(num+1);
    if (blocos.contains((Integer) num))
        return 'If(c1,c2,c3,condicao,c4,c5,inst2,inst1);
}
While(c1,c2,c3,condicao,c4,c5,inst,c6) -> {
    int num = (Integer) numInstrucao.remove((int) 0);
    numInstrucao.add(num+1);
    if (blocos.contains((Integer) num))
        return 'While(c1,c2,c3,Nao(condicao),c4,c5,inst,c6);
}
For(c1,c2,decl,c3,condicao,c4,c5,exp,c6,c7,inst,c8) -> {
    int num = (Integer) numInstrucao.remove((int) 0);
    numInstrucao.add(num+1);
    if (blocos.contains((Integer) num))
        return 'For(c1,c2,decl,c3,Nao(condicao),c4,c5,exp,c6,c7,
            inst,c8);
}
i@Return(_,_,exp,_) -> {
    int num = (Integer) numInstrucao.remove((int) 0);
    numInstrucao.add(num+1);
}
}
visit Expressao {
e@ExpNum(exp1,_,_,op,_,_,exp2) -> {
    int num = (Integer) numInstrucao.remove((int) 0);
    numInstrucao.add(num+1);
}
e@Ou(cond1,_,_,cond2) -> {
    int num = (Integer) numInstrucao.remove((int) 0);
    numInstrucao.add(num+1);
}
e@E(cond1,_,_,cond2) -> {
    int num = (Integer) numInstrucao.remove((int) 0);
    numInstrucao.add(num+1);
}
e@Comp(exp1,_,_,opComp,_,_,exp2) -> {
    int num = (Integer) numInstrucao.remove((int) 0);
    numInstrucao.add(num+1);
}
```



```
    }
    }
}

private String compileAnnot(Instrucao inst) {
    NumToInt numInstrucao = new NumToInt(1);
    String toReturn = compileAnnotInstrucao(inst, numInstrucao);
    return toReturn.concat("Halt");
}

private String compileAnnotInstrucao(Instrucao i, NumToInt
numInstrucao) {
    %match(i) {
        Atribuicao(_, id, _, opAtrib, _, exp, _) -> {
            String genExp = 'compileAnnotExpressoes(exp, numInstrucao);
            String prefix;
            if (actualFunctionName.equals(""))
                prefix = "";
            else
                prefix = actualFunctionName + "_";

            %match(opAtrib) {
                Atrib() -> { return "Pusha_" + prefix + 'id + "\",' +
                    genExp + "Store,"; }
                Mult() -> { return "Pusha_" + prefix + 'id + "\',Pusha_" +
                    prefix + 'id + "\',Load," + genExp + "Mul,Store,"; }
                Div() -> { return "Pusha_" + prefix + 'id + "\',Pusha_" +
                    prefix + 'id + "\',Load," + genExp + "Div,Store,"; }
                Soma() -> { return "Pusha_" + prefix + 'id + "\',Pusha_" +
                    prefix + 'id + "\',Load," + genExp + "Add,Store,"; }
                Sub() -> { return "Pusha_" + prefix + 'id + "\',Pusha_" +
                    prefix + 'id + "\',Load," + genExp + "Sub,Store,"; }
            }
            return "";
        }
    }

    Declaracao(_, tipo, _, decls, _, _) -> {
        String genDecl = 'compileAnnotDeclaracoes(decls, tipo,
            numInstrucao);
        functionsDeclarations.append(genDecl);
        return "";
    }

    If(_, _, _, condicao, _, _, inst1, inst2) -> {
        String genCondicao = 'compileAnnotExpressoes(condicao,
            numInstrucao);
        String genInst1 = 'compileAnnotInstrucao(inst1, numInstrucao)
            ;
    }
}
```



```
String genInst2 = 'compileAnnotInstrucao(inst2 , numInstrucao)
;
int num = numInstrucao.inc();

return genCondicao + "Jumpf_\\"senao" + num + "\\" , " + genInst1
+ "Jump_\\"fse" + num + "\\" , ALabel_\\"senao" + num + "\\" , "
+ genInst2 + "ALabel_\\"fse" + num + "\\" , " ;
}

While(_,_,_, condicao ,_,_, inst ,_) -> {
String genCondicao = 'compileAnnotExpressoes(condicao ,
numInstrucao);
String genInst = 'compileAnnotInstrucao(inst , numInstrucao);
int num = numInstrucao.inc();

return genCondicao + "Jumpf_\\"fenq" + num + "\\" , " + genInst +
"ALabel_\\"fenq" + num + "\\" , " ;
}

For(_,_, decl ,_, condicao ,_,_, exp ,_,_, inst ,_) -> {
String genDecl = 'compileAnnotInstrucao(decl , numInstrucao);
String genCondicao = 'compileAnnotExpressoes(condicao ,
numInstrucao);
String genExp = 'compileAnnotExpressoes(exp , numInstrucao);
String genInst = 'compileAnnotInstrucao(inst , numInstrucao);

functionsDeclarations.append(genDecl);

return genCondicao.concat(genExp).concat(genInst);
}

Return(_,_, exp ,_) -> {
String genExp = 'compileAnnotExpressoes(exp , numInstrucao);
String prefix = "f:";
String ret = "Ret,";
String storeVarFunc = "Pusha_\\" + prefix +
actualFunctionName + "\\" , " + genExp + "Store,";

return storeVarFunc;
}

Funcao(_, tipo ,_, nome ,_,_, argumentos ,_,_, inst ,_) -> {
int actualMemAddress = memAdress;
memAdress++;
int sizeAddress = 1;

actualFunctionName = 'nome;
String prefix = "f:";
```



```
String functionDeclaration = "Decl_\\" + prefix + 'nome + "\\"  
    \" + actualMemAddress + "\" + sizeAddress + \",\";  
String functionRet = \"\";  
%match(tipo) {  
    DVoid() -> { if (!actualFunctionName.equals(\"main\"))  
        functionRet = \"Ret,\"; }  
    _ -> { functionRet = \"\"; }  
}  
String halt = actualFunctionName.equals(\"main\") ? \"Halt,\" : \"  
    \";  
String genArgs = 'compileArguments(nome, argumentos);  
  
functionsDeclarations.append(functionDeclaration);  
functionsDeclarations.append(genArgs);  
  
String genInst = 'compileAnnotInstrucao(inst, numInstrucao);  
String function = \"ALabel_\\"f:\" + 'nome + "\\",\" + genInst +  
    functionRet + halt;  
  
    return function;  
}  
  
Exp(exp) -> {  
    callReturnNeeded = false;  
    String exp = 'compileAnnotExpressoes(exp, numInstrucao);  
    callReturnNeeded = true;  
  
    return exp;  
}  
  
SeqInstrucao(inst1, inst*) -> {  
    String genInst = 'compileAnnotInstrucao(inst1, numInstrucao);  
    String seqInst = genInst.concat('compileAnnotInstrucao(inst*,  
        numInstrucao));  
  
    return seqInst;  
}  
}  
return \"\";  
}  
  
private String compileArguments(String functionName, Argumentos  
    args) {  
    %match(args) {  
        ListaArgumentos(arg1, tailArg*) -> {  
            return compileArguments(functionName, 'arg1) +  
                compileArguments(functionName, 'tailArg);  
        }  
        Argumento(_,_,_,idArg,_) -> {
```




```
DChar() -> { return "Decl_" + prefix + 'id + "\"' +
    actualMemAddress + "_" + sizeAddress + "," + storeValue
; }
DBoolean() -> { return "Decl_" + prefix + 'id + "\"' +
    actualMemAddress + "_" + sizeAddress + "," + storeValue
; }
DFloat() -> { return "Decl_" + prefix + 'id + "\"' +
    actualMemAddress + "_" + sizeAddress + "," + storeValue
; }
DVoid() -> { return "Decl_" + prefix + 'id + "\"' +
    actualMemAddress + "_" + sizeAddress + "," + storeValue
; }
}
return "";
}
}
return "";
}

private String compileAnnotExpressoes(Expressao e, NumToInt
numInstrucao) {
    %match(e) {
        ExpNum(exp1,_,op,_,exp2) -> {
            String genExp1 = 'compileAnnotExpressoes(exp1, numInstrucao);
            String genExp2 = 'compileAnnotExpressoes(exp2, numInstrucao);

            %match(op) {
                Mais() -> { return genExp1 + genExp2 + "Add,"; }
                Vezes() -> { return genExp1 + genExp2 + "Mul,"; }
                Divide() -> { return genExp1 + genExp2 + "Div,"; }
                Menos() -> { return genExp1 + genExp2 + "Sub,"; }
                Mod() -> { return genExp1.concat(genExp2); }
            }
            return "";
        }
    }

    Id(id) -> {
        String prefix;
        if (actualFunctionName.equals(""))
            prefix = "";
        else
            prefix = actualFunctionName + "_";

        return "Pusha_" + prefix + 'id + "\"',Load,";
    }

    Pos(exp) -> { return 'compileAnnotExpressoes(exp, numInstrucao)
; }
```



```
Neg(exp) -> { return 'compileAnnotExpressoes(exp, numInstrucao)
; }

Nao(exp) -> {
    String genExp = 'compileAnnotExpressoes(exp, numInstrucao);
    return genExp + "Not,";
}

Call(_,id,_,_,parametros,_,_) -> {
    Argumentos argumentos = functionSignatures.get('id);
    String prefix = "f:";
    String loadReturn = callReturnNeeded ? "Pusha_\\"" + prefix
        + 'id + "\",Load," : ";
    String genCallParameters = compileCallParameters('id,
        argumentos, 'parametros, numInstrucao);
    String call = "Call_\\"" + prefix + 'id + "\",";
    return genCallParameters + call + loadReturn;
}

IncAntes(opInc,id) -> {
    String prefix;
    if (actualFunctionName.equals(""))
        prefix = "";
    else
        prefix = actualFunctionName + "_";

    %match(opInc) {
        Inc() -> { return "Pusha_\\"" + prefix + 'id + "\",Inc,Pusha
            _\\"" + prefix + 'id + "\",Load,"; }
        Dec() -> { return "Pusha_\\"" + prefix + 'id + "\",Dec,Pusha
            _\\"" + prefix + 'id + "\",Load,"; }
    }
    return 'id;
}

IncDepois(opInc,id) -> {
    String prefix;
    if (actualFunctionName.equals(""))
        prefix = "";
    else
        prefix = actualFunctionName + "_";

    %match(opInc) {
        Inc() -> { return "Pusha_\\"" + prefix + 'id + "\",Load,
            Pusha_\\"" + prefix + 'id + "\",Inc,"; }
        Dec() -> { return "Pusha_\\"" + prefix + 'id + "\",Load,
            Pusha_\\"" + prefix + 'id + "\",Dec,"; }
    }
    return 'id;
```



```
}

Condicional(condicao,_,_,exp1,_,_,exp2) -> {
    String genCondicao = 'compileAnnotExpressoes(condicao,
        numInstrucao);
    String genExp1 = 'compileAnnotExpressoes(exp1, numInstrucao);
    String genExp2 = 'compileAnnotExpressoes(exp2, numInstrucao);

    return genCondicao.concat(genExp1).concat(genExp2);
}

Int(i) -> { return "Pushi_" + 'i + ","; }

Char(c) -> { return "Pushc_" + 'c.charAt(0) + "','; }

True() -> { return "Pushb_true,"; }

False() -> { return "Pushb_false,"; }

Float(f) -> { return "Pushf_" + 'f + ","; }

Ou(cond1,_,_,cond2) -> {
    String genCond1 = 'compileAnnotExpressoes(cond1, numInstrucao
    );
    String genCond2 = 'compileAnnotExpressoes(cond2, numInstrucao
    );

    return genCond1 + genCond2 + "Or,";
}

E(cond1,_,_,cond2) -> {
    String genCond1 = 'compileAnnotExpressoes(cond1, numInstrucao
    );
    String genCond2 = 'compileAnnotExpressoes(cond2, numInstrucao
    );

    return genCond1 + genCond2 + "And,";
}

Comp(exp1,_,_,opComp,_,_,exp2) -> {
    String genExp1 = 'compileAnnotExpressoes(exp1, numInstrucao);
    String genExp2 = 'compileAnnotExpressoes(exp2, numInstrucao);

    %match(opComp) {
        Maior() -> { return genExp1 + genExp2 + "Gt,"; }
        Menor() -> { return genExp1 + genExp2 + "Lt,"; }
        MaiorQ() -> { return genExp1 + genExp2 + "GoEq,"; }
        MenorQ() -> { return genExp1 + genExp2 + "LoEq,"; }
        Dif() -> { return genExp1 + genExp2 + "Neq,"; }
    }
```



```
    Igual() -> { return genExp1 + genExp2 + "Eq,"; }
  }
}

Input(_,_,_,tipo,_,_) -> {
  %match(tipo) {
    DInt() -> { return "IIn_int,"; }
    DChar() -> { return "IIn_char,"; }
    DBoolean() -> { return "IIn_boolean,"; }
    DFloat() -> { return "IIn_float,"; }
  }
}

Print(_,_,_,exp,_,_) -> {
  String genExp = 'compileAnnotExpressoes(exp, numInstrucao);

  return genExp + "IOut,";
}

Expressoes(exp1, exp*) -> {
  String genExp = 'compileAnnotExpressoes(exp1, numInstrucao);
  String exps = genExp.concat('compileAnnotExpressoes(exp*,
    numInstrucao));

  return exps;
}

Empty() -> { return ""; }
}
return "";
}

private String compileCallParameters(String functionName,
  Argumentos argumentos, Parametros parametros, NumToInt
  numInstrucao) {
  %match (parametros, argumentos){
    ListaParametros(param1,tailParam*), ListaArgumentos(arg1,
    tailArg*) -> {
      return compileCallParameters(functionName, 'arg1, 'param1,
        numInstrucao) + compileCallParameters(functionName, '
        tailArg, 'tailParam, numInstrucao);
    }
  }
  Parametro(_,exp,_), Argumento(_,_,_,idArg,_) -> {
    String genExp = 'compileAnnotExpressoes(exp, numInstrucao);
    String prefix = functionName + "_";
    return "Pusha_\\"" + prefix + 'idArg + "\",' + genExp + "Store
    ,";
  }
}
}
```



```
        return "";
    }

    private String genAnnotation(int i) {
        if (i == 1) {
            return "Pushi_" + i + ",IOut,";
        }
        else {
            return "Pushc_'',IOut,Pushi_" + i + ",IOut,";
        }
    }

    private static boolean parseFile(String filename, TreeSet<Integer>
        blocos) {
        try {
            BufferedReader br = new BufferedReader( new FileReader(filename
                ));
            String line = "";
            StringTokenizer token = null;

            while((line = br.readLine()) != null) {
                token = new StringTokenizer(line, ",");

                while(token.hasMoreTokens()) {
                    String tokenS = token.nextToken();
                    blocos.add(Integer.parseInt(tokenS));
                }
            }

            return true;
        } catch(Exception e) {
            return false;
        }
    }
}

class NumToInt{
    private int num;

    public NumToInt(int num) {
        this.num = num;
    }

    public NumToInt(){
        num = 0;
    }

    public int inc(){
        return num++;
    }
}
```



```
    }  
  
    public int get() {  
        return num;  
    }  
}
```



K Código Gom Máquina Virtual v2

```
module maqv.msp
imports int String
abstract syntax

Instrucoes = Instrucoes(Instrucao*)

Instrucao = ALabel(id:String)
           | Call(id:String)
           | Ret()
           | Add()
           | Sub()
           | Div()
           | Mul()
           | Mod()
           | Inc()
           | Dec()
           | Eq()
           | Neq()
           | Gt()
           | GoEq()
           | Lt()
           | LoEq()
           | Nott()
           | Or()
           | And()
           | Halt()
           | IIn(tipo:DefTipo)
           | IOut()
           | Jump(id:String)
           | Jumpf(id:String)
           | Push(t:Termo)
           | Pusha(t:Termo)
           | Load()
           | Store()
           | Decl(id:String,initMemAddress:int, size:int)

DefTipo = DInt() | DChar() | DBoolean() | DFloat()

Boool = True() | False()

Stackk = Stackk(Termo*)

Termo = I(i:int)
       | S(id:String)
       | B(b:Boool)
       | F(f:int)
       | Vazio()
```



L Código Main Máquina Virtual v2

```
package maqv;

import maqv.msp.mspAdaptor;
import maqv.msp.types.*;
import org.antlr.runtime.CommonTokenStream;
import org.antlr.runtime.ANTLRInputStream;
import org.antlr.runtime.tree.Tree;
import tom.library.utils.Viewer;
import tom.library.sl.*;
import java.util.*;
import java.lang.*;
import java.io.*;

public class Main {
    %include{sl.tom}
    %include{util/HashMap.tom}
    %include{util/types/Collection.tom}
    %include{../genMaqV/maqv/msp/msp.tom}

    private Instrucoes programa;
    private Stackk stack;
    private Map<String,Integer> symbols;
    private ArrayList<Termo> heap;
    private int pc;
    private int numProg;
    private StringBuilder output;

    public static void main(String[] args) {
        try {
            mspLexer lexer = new mspLexer(new ANTLRInputStream(new
                FileInputStream(args[0])));
            CommonTokenStream tokens = new CommonTokenStream(lexer);
            mspParser parser = new mspParser(tokens);
            // Parse the input expression
            Tree b = (Tree) parser.programa().getTree();
            //System.out.println("Result = " + mspAdaptor.getTerm(b)); //
                name of the Gom module + Adaptor
            Instrucoes p = (Instrucoes) mspAdaptor.getTerm(b);

            Main main = new Main(p);

            main.run(p);

            /* Export this representation to .dot file*/
            /*
            try{
```




```
        FileWriter out=new FileWriter("gram.dot");
        Viewer.toDot(p,out);
    }
    catch (IOException e){
        System.out.println("ERROR in dot file");
    }
}
*/
if (args.length > 1) {
    try {
        PrintWriter pw = new PrintWriter(new BufferedWriter(new
            FileWriter(args[1], true)));
        pw.print(main.getOutput());
        pw.flush(); pw.close();
    }
    catch (IOException e){
        System.err.println("exception:_" + e);
        return;
    }
}
else {
    System.out.println(main.getOutput());
}
} catch (Exception e) {
    e.printStackTrace();
}
}

public Main(Instrucoes insts) {
    programa = insts;
    stack = 'Stackk();
    heap = new ArrayList<Termo>();
    symbols = new HashMap<String , Integer>();
    pc = 0;
    numProg = 0;
    output = new StringBuilder();
}

public String getOutput(){
    return output.toString();
}

private Instrucoes getNInstr(Instrucoes prog, int progc){
    %match (prog){
        i@Instrucoes(inst ,insts*) -> {
            if (progc == 0) {
                return 'i;
            }
            else { return 'getNInstr(insts*,progc-1); }
        }
    }
}
```



```
    }
    return 'Instrucoes()';
}

private Instrucoes jmp(Instrucoes prog, String label){
    %match (prog){
        Instrucoes(ALabel(l), insts*) -> {
            if (label.equals('l')) { return 'insts*'; }
            else { return 'jmp(insts*,label); }
        }
        Instrucoes(_, insts*) -> { return 'jmp(insts*,label); }
    }
    return 'Instrucoes()';
}

private void pushStack(Termo termo){
    %match (stack){
        Stackk(terms*) -> { this.stack = 'Stackk(termo,terms*); }
    }
}

private void popStack(){
    %match (stack){
        Stackk(termo1, terms*) -> { this.stack = 'Stackk(terms*); }
    }
}

private Termo topStack() {
    %match(stack){
        Stackk(termo, terms*) -> { return 'termo'; }
    }
    return 'Vazio()';
}

private void memAlloc(String symbol, int initMemAddress, int size)
{
    Integer memAddress = symbols.get(symbol);
    if (memAddress == null) {
        symbols.put(symbol, initMemAddress);
        for (int i = 0; i < size; i++){
            heap.add(initMemAddress + i, 'Vazio()');
        }
    }
}

private int getMemAddress(String symbol) {
    return symbols.get(symbol);
}
```



```
private Termino getMem(int memAddress) {
    return heap.get(memAddress);
}

private void putMem(int memAddress, Termino value) {
    heap.set(memAddress, value);
}

public String run(Instrucoes prog) {
    pc++;
    %match (prog){
        Instrucoes(inst, instrs*) -> {
            %match(inst) {
                ALabel(id) -> { return 'run(instrs*); }
                Call(id) -> {
                    'pushStack(I(pc));
                    prog = 'jmp(prog, id);
                    return 'run(prog);
                }
                Ret() -> {
                    Termino progCount = 'topStack();
                    'popStack();
                    %match(progCount) {
                        I(valor) -> {
                            pc = 'valor;
                            prog = 'getNInstr(programa, valor);
                        }
                    }
                    return 'run(prog);
                }
            }
        }
        Add() -> {
            %match (stack){
                Stackk(I(v2), I(v1), resto*) -> {
                    stack = 'resto*;
                    int resultado = 'v1+'v2;
                    'pushStack(I(resultado));
                    return 'run(instrs*);
                }
            }
        }
        Sub() -> {
            %match (stack){
                Stackk(I(v2), I(v1), resto*) -> {
                    stack = 'resto*;
                    int resultado = 'v1 - 'v2;
                    'pushStack(I(resultado));
                    return 'run(instrs*);
                }
            }
        }
    }
}
```



```
}
Div() -> {
    %match (stack){
        Stackk(I(v2),I(v1),resto*) -> {
            stack = 'resto*';
            int resultado = 'v1 / 'v2;
            'pushStack(I(resultado));
            return 'run(instrs*);
        }
    }
}
Mul() -> {
    %match (stack){
        Stackk(I(v2),I(v1),resto*) -> {
            stack = 'resto*';
            int resultado = 'v1 * 'v2;
            'pushStack(I(resultado));
            return 'run(instrs*);
        }
    }
}
Mod() -> {
    %match (stack){
        Stackk(I(v2),I(v1),resto*) -> {
            stack = 'resto*';
            int resultado = 'v1 % 'v2;
            'pushStack(I(resultado));
            return 'run(instrs*);
        }
    }
}
Inc() -> {
    Termo t = 'topStack();
    'popStack();
    %match(t) {
        I(memAddress) -> {
            Termo t2 = getMem('memAddress);
            int valorToInc = 0;
            %match(t2) {
                I(v1) -> { valorToInc = 'v1+1; }
            }
            putMem('memAddress,'I(valorToInc));

            return 'run(instrs*);
        }
    }
    return 'run(instrs*);
}
Dec() -> {
```



```
Termo t = 'topStack();
'popStack();
%match(t) {
  I(memAddress) -> {
    Termo t2 = getMem('memAddress);
    int valorToDec = 0;
    %match(t2) {
      I(v1) -> { valorToDec = 'v1-1; }
    }
    putMem('memAddress, 'I(valorToDec));

    return 'run(instrs*);
  }
}
return 'run(instrs*);
}
Eq() -> {
  %match (stack){
    Stackk(I(v2),I(v1),resto*) -> {
      stack = 'resto*;
      Bool resultado = ('v1 == 'v2) ? 'True() : 'False();
      'pushStack(B(resultado));
      return 'run(instrs*);
    }
  }
}
Neq() -> {
  %match (stack){
    Stackk(I(v2),I(v1),resto*) -> {
      stack = 'resto*;
      Bool resultado = ('v1 == 'v2) ? 'False() : 'True();
      'pushStack(B(resultado));
      return 'run(instrs*);
    }
  }
}
Gt() -> {
  %match (stack){
    Stackk(I(v2),I(v1),resto*) -> {
      stack = 'resto*;
      Bool resultado = ('v1 > 'v2) ? 'True() : 'False();
      'pushStack(B(resultado));
      return 'run(instrs*);
    }
  }
}
GoEq() -> {
  %match (stack){
    Stackk(I(v2),I(v1),resto*) -> {
```



```
        stack = 'resto*;  
        Bool resultado = ('v1 >= 'v2) ? 'True() : 'False();  
        'pushStack(B(resultado));  
        return 'run(instrs*);  
    }  
}  
}  
Lt() -> {  
    %match (stack){  
        Stackk(I(v2),I(v1),resto*) -> {  
            stack = 'resto*;  
            Bool resultado = ('v1 < 'v2) ? 'True() : 'False();  
            'pushStack(B(resultado));  
            return 'run(instrs*);  
        }  
    }  
}  
LoEq() -> {  
    %match (stack){  
        Stackk(I(v2),I(v1),resto*) -> {  
            stack = 'resto*;  
            Bool resultado = ('v1 <='v2) ? 'True() : 'False();  
            'pushStack(B(resultado));  
            return 'run(instrs*);  
        }  
    }  
}  
Nott() -> {  
    Termo t = 'topStack();  
    'popStack();  
    %match(t) {  
        i@I(v1) -> { 'pushStack(i); }  
        s@S(v1) -> { 'pushStack(s); }  
        f@F(v1) -> { 'pushStack(f); }  
        B(True()) -> { 'pushStack(B(False())); }  
        B(False()) -> { 'pushStack(B(True())); }  
    }  
    return 'run(instrs*);  
}  
Or() -> {  
    %match (stack){  
        Stackk(B(v2),B(v1),resto*) -> {  
            stack = 'resto*;  
            boolean valor1 = true, valor2 = true;  
            %match (v1){  
                True() -> { 'valor1 = true; }  
                False() -> { 'valor1 = false; }  
            }  
            %match (v2){
```



```
        True() -> { 'valor2 = true; }
        False() -> { 'valor2 = false; }
    }
    Bool resultado = (valor1 || valor2) ? 'True() : '
        False();
    'pushStack(B(resultado));
    return 'run(instrs*);
}
}
}
And() -> {
    %match (stack){
        Stackk(B(v2),B(v1),resto*) -> {
            stack = 'resto*;
            boolean valor1 = true, valor2 = true;
            %match (v1){
                True() -> { 'valor1 = true; }
                False() -> { 'valor1 = false; }
            }
            %match (v2){
                True() -> { 'valor2 = true; }
                False() -> { 'valor2 = false; }
            }
            Bool resultado = (valor1 && valor2) ? 'True() : '
                False();
            'pushStack(B(resultado));
            return 'run(instrs*);
        }
    }
}
Halt() -> { return ""; }
IIn(tipo) -> {
    BufferedReader br = new BufferedReader(new
        InputStreamReader(System.in));
    try{
        String iin = br.readLine();
        %match(tipo) {
            DInt() -> {
                try{
                    int v1 = Integer.parseInt(iin);
                    'pushStack(I(v1));
                }catch(NumberFormatException nfe){
                    System.err.println("Invalid Format!");
                }
            }
            DChar() -> {
                String s = ""+iin.charAt(0);
                'pushStack(S(s));
            }
        }
    }
}
```





```
}
Push(t) -> {
    'pushStack(t);
    return 'run(instrs*);
}
Pusha(t) -> {
    %match(t) {
        S(id) -> {
            int memAddress = getMemAddress('id);
            'pushStack(I(memAddress));

            return 'run(instrs*);
        }
    }
    return 'run(instrs*);
}
Load() -> {
    Termo t = 'topStack();
    'popStack();
    %match(t) {
        I(memAddress) -> {
            Termo t2 = getMem('memAddress);
            'pushStack(t2);

            return 'run(instrs*);
        }
    }
    return 'run(instrs*);
}
Store() -> {
    Termo t = 'topStack();
    'popStack();
    Termo t2 = 'topStack();
    'popStack();
    %match(t2) {
        I(memAddress) -> {
            putMem('memAddress,t);

            return 'run(instrs*);
        }
    }
    return 'run(instrs*);
}
Decl(id,initMemAddress,size) -> {
    memAlloc('id,'initMemAddress,'size);
    return 'run(instrs*);
}
}
```



```
    }  
    return "";  
  }  
}
```



M Código java da aplicação de detecção de falhas e análise de injeção de falhas

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package ats;

/**
 *
 * @author luis
 */
public class Bc {
    private Integer boco;
    private Integer vezes;

    public Bc(Integer boco) {
        this.boco = boco;
        this.vezes = 0;
    }

    public Bc(Integer boco, Integer vezes) {
        this.boco = boco;
        this.vezes = vezes;
    }

    public Integer getBoco() {
        return boco;
    }

    public void setBoco(Integer boco) {
        this.boco = boco;
    }

    public Integer getVezes() {
        return vezes;
    }

    public void setVezes(Integer vezes) {
        this.vezes = vezes;
    }

    void incVezes() {
        this.vezes++;
    }

    @Override
```



```
        public String toString() {
            StringBuilder s = new StringBuilder();
            s.append("O_bloco_de_codigo_");
            s.append(boco);
            s.append(",_foi_utilizado_");
            s.append(vezes);
            s.append("_vezes.");
            s.append("\n");
            return s.toString();
        }

    }

    /*
     * To change this template, choose Tools | Templates
     * and open the template in the editor.
     */
    package ats;

    import java.util.ArrayList;

    /**
     *
     * @author luis
     */
    public class coiso {
        private ArrayList <ArrayList<Integer>> matriz;
        private ArrayList<Integer> res;
        private ArrayList<Float> prob;

        public coiso(ArrayList<ArrayList<Integer>> matriz , ArrayList<
            Integer> res) {
            this.matriz = matriz;
            this.res = res;
            this.prob = probabilidades(matriz , res);
        }

        // calcula probabilidade de todos os blocos de codigo serem os
        // responsaveis pelo comportamento errado
        public ArrayList<Float> probabilidades (ArrayList <ArrayList<
            Integer>> matris , ArrayList<Integer> res){
            ArrayList<Float> probs = new ArrayList<Float>();
            float aux;
            int n11 = 0;
            int n10 = 0;
            int n01 = 0;

            for (int i = 0; i < matris.get(0).size() ; i++ ){
```



```
        for(int j = 0; j < res.size() ;j++){
            int a = matris.get(j).get(i);
            int b = res.get(j);
            if( a == 1 && b == 1 ) {
                n11++;
            }
            else if ( a == 1 && b == 0 ) {
                n10++;
            }
            else if ( a == 0 && b == 1 ) {
                n01++;
            }
        }
        aux = (n11 + n10 + n01);
        aux = (n11 /aux);
        probs.add(aux);
        n11 = 0;
        n10 = 0;
        n01 = 0;
    }

    return probs;
}

public ArrayList<ArrayList<Integer>> getMatriz() {
    return matriz;
}

public void setMatriz(ArrayList<ArrayList<Integer>> matriz) {
    this.matriz = matriz;
}

public ArrayList<Integer> getRes() {
    return res;
}

public void setRes(ArrayList<Integer> res) {
    this.res = res;
}

public ArrayList<Float> getProb() {
    return prob;
}

public void setProb(ArrayList<Float> prob) {
    this.prob = prob;
}
```



```
}

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package ats;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.StringTokenizer;
import java.util.TreeSet;
import javax.swing.text.html.HTMLDocument;

/**
 *
 * @author luis
 */

public class DadosEntrada {

    private Integer max =0;
    private ArrayList <String> resultadosEsperados = new ArrayList <
        String>();
    private ArrayList <Bc> bCodigo = new ArrayList <Bc>();
    private FecheiroTeste normal = new FecheiroTeste();
    private FecheiroTeste falhas = new FecheiroTeste();

    public ArrayList<Bc> bRanking(){
        return quicksort(bCodigo);
    }

    private ArrayList<Bc> quicksort(ArrayList<Bc> input){
        if(input.size() <= 1){
            return input;
        }
        int middle = (int) Math.ceil((double)input.size() / 2);
        Bc pivot = input.get(middle);

        ArrayList<Bc> less = new ArrayList<Bc>();
        ArrayList<Bc> greater = new ArrayList<Bc>();
        for (int i = 0; i < input.size(); i++) {
```



```
        if(input.get(i).getVezes() <= pivot.getVezes()){
            if(i == middle){
                continue;
            }
            less.add(input.get(i));
        }
        else{
            greater.add(input.get(i));
        }
    }
    return concatenate(quicksort(less), pivot, quicksort(greater)
    );
}
private ArrayList<Bc> concatenate(ArrayList<Bc> less, Bc pivot,
    ArrayList<Bc> greater){
    ArrayList<Bc> list = new ArrayList<Bc>();

    for (int i = 0; i < greater.size(); i++) {
        list.add(greater.get(i));
    }

    list.add(pivot);

    for (int i = 0; i < less.size(); i++) {
        list.add(less.get(i));
    }
    return list;
}
private void addBcodigo(Integer bc){

    int tam = this.bCodigo.size();
    if(bc <= tam){
        bCodigo.get((bc - 1)).incVezes();
    }
    else{
        while((bc-1) != this.bCodigo.size()){
            if(tam > 0){
                Integer last = bCodigo.get((bCodigo.size()-1)).
                    getBoco();
                bCodigo.add(new Bc((last+1)));
            }
            else{
                bCodigo.add(new Bc(1));
            }
        }
        bCodigo.add(new Bc(bc, 1));
    }
}
```



```
public ArrayList <Integer> blocosNutilizados() {

    ArrayList <Integer> notUsed = new ArrayList<Integer>();

    for(int i =0; i < this.bCodigo.size(); i++){
        if(this.bCodigo.get(i).getVezes() == 0) {
            notUsed.add((i+1));
        }
    }

    return notUsed;
}

// da maior bloco de codigo encontrado
public Integer getMax() {
    return max;
}

public ArrayList<Bc> getbCodigo() {
    return bCodigo;
}
/**
 * @param args the command line arguments
 */

// carrega de um ficheiro a informacao relativa aos resultados
// dos testes
// chamado com o valor 1 o ficheiro e' o normal
// chamado com o valor 2 e' o ficheiro com as falhas

public boolean lerFecheiroTeste(String file ,int i) throws
FileNotFoundException, IOException{
    if(i == 1){
        this.normal.clear();
    }
    else if ( i== 2){
        this.falhas.clear();
    }
    boolean t = true;
    BufferedReader br = new BufferedReader(new FileReader(file));

    while(br.ready() && t){
        String linha = br.readLine();
        if(i == 1){
            this.normal.getLinhas().add(linha);
        }
        else if ( i== 2){
            this.falhas.getLinhas().add(linha);
        }
    }
}
```




```
        }
        t = partir(linha,i);
    }
    br.close();

    if ( i== 1 && bCodigo.size()>this.max){
        t = false;
    }else if (i== 1 && bCodigo.size()>this.max){
        this.addBcodigo(this.max);
        this.bCodigo.get(this.max-1).setVezes(0);
    }

    return t;
}
// carrega os resultados esperados de um ficheiro para memoria
public void lerFicheiroRes(String file) throws
    FileNotFoundException, IOException{
    this.resltadosEsperados.clear();
    BufferedReader br = new BufferedReader(new FileReader(file));
    while(br.ready()){
        String linha = br.readLine();
        this.resltadosEsperados.add(linha);
    }
    br.close();
}

// ver se os testes cobrem todos os blocos de codigo
public ArrayList <Integer> testCoverage(){

    return this.normal.semEfeito(falhas);
}

// verifica se o ficheiro de teste e de resultados esperads tem
// os mesmo numero de elemntos.
public boolean tamanho( int i){

    boolean aux = false;
    if(i == 1)
        aux = tamanho(this.normal);
    else if ( i == 2)
        aux=tamanho(this.falhas);

    return aux;
}
private boolean tamanho( FechoiroTeste ft){
    boolean aux;
    if(ft.getDados().size() == resltadosEsperados.size())
        aux = true;
}
```



```
        else
            aux=false;

        return aux;

    }
    // utilizado no mentudo lerFecheiroTeste, parte uma linha do
    // ficheiro de testes
    // e coloca os seu elemtos nos seu siteo correcto, tambem vai
    // validando o ficheiro de dados.
    public boolean partir(String linha, int j){
        boolean t = true;
        TreeSet<Integer> aux = new TreeSet<Integer>();
        StringTokenizer bc = new StringTokenizer(linha, "#");
        if(bc.hasMoreElements()){
            try {
                int max = Integer.parseInt(bc.nextToken());
                if(this.max > 0 && this.max != max){
                    t = false;
                }
            }
            else{
                this.max=max;
            }
        } catch(NumberFormatException nfe) {
            t = false;
        }
    }
    else{
        t = false;
    }
    if(bc.hasMoreElements() && t){
        StringTokenizer str = new StringTokenizer(bc.nextToken(), ";");
        ;
        String blocos = str.nextToken();
        if(str.hasMoreElements()){
            StringTokenizer str2 = new StringTokenizer(blocos, ",");
            Integer i = 0;
            while(str2.hasMoreElements() && t){
                try {
                    i = Integer.parseInt(str2.nextToken());
                    this.addBcodigo(i);
                } catch(NumberFormatException nfe) {
                    t = false;
                }
            }
            if(t){
                aux.add(i);
            }
        }
    }
}
```



```
        if(j == 1){
            this.normal.getResltados().add(str.nextToken());
            this.normal.getDados().add(aux);
        }
        else if (j == 2){
            this.falhas.getResltados().add(str.nextToken());
            this.falhas.getDados().add(aux);
        }
    }
    else{
        t = false;
    }
}
else{
    t = false;
}
return t;
}

// inicia a matriz utilizada no algoritmos SFL

private ArrayList <ArrayList<Integer>> initMatris (int i){
    ArrayList <ArrayList<Integer>> tablea = new ArrayList <
        ArrayList<Integer>>();

    if (i==1){
        tablea=initMatris(this.normal);
    }
    else if( i ==2 ){
        tablea=initMatris(this.falhas);
    }

    return tablea;
}

private ArrayList <ArrayList<Integer>> initMatris (FecheiroTeste
ft){
    ArrayList <ArrayList<Integer>> tablea = new ArrayList <
        ArrayList<Integer>>();

    for(int i =0; i< ft.getDados().size(); i++){
        ArrayList<Integer> linhaTab = new ArrayList<Integer>();
        for (int j=0; j < this.max; j++){
            linhaTab.add(0);
        }
        tablea.add(linhaTab);
    }

    return tablea;
}
```



```
}

// constroi a matris com os dados dos ficheiros ja carregados que
// estao nas es truturas de dados
// reservadas para esse efeito
public ArrayList <ArrayList<Integer>> matris (int i){
    ArrayList <ArrayList<Integer>> tablea = initMatris(i);

    if (i==1){
        tablea=matris(this.normal,i);
    }
    else if( i ==2 ){
        tablea=matris(this.falhas,i);
    }

    return tablea;
}
private ArrayList <ArrayList<Integer>> matris (FicheiroTeste ft ,
int j){
    ArrayList <ArrayList<Integer>> tablea = initMatris(j);

    Iterator it1 = ft.getDados().iterator();
    int i=0;
    while (it1.hasNext())
    {
        TreeSet<Integer> aux = (TreeSet<Integer>) it1.next();
        Iterator it = aux.iterator();
        while (it.hasNext()){
            tablea.get(i).set((Integer)it.next()-1, 1);
        }
        i++;
    }

    return tablea;
}

// verefica o resultado obetido no teste coincide com o esperado
// e coloca essa
// avaliacao um vector de resultados

public ArrayList<Integer> res (int i){
    ArrayList<Integer> aux = new ArrayList<Integer>();
    if (i==1){
        aux=res(this.normal);
    }
    else if( i ==2 ){
        aux=res(this.falhas);
    }
    return aux;
}
```



```
}
private ArrayList<Integer> res (FecheiroTeste ft){
    ArrayList<Integer> aux = new ArrayList<Integer>();

    Iterator it1 = ft.getResltados().iterator();
    Iterator it2 = resultadosEsperados.iterator();

    while(it1.hasNext() && it2.hasNext()){
        if(it1.next().equals(it2.next()))
            aux.add(0);
        else
            aux.add(1);
    }

    return aux;
}
}

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package ats;

import java.util.ArrayList;
import java.util.TreeSet;

/**
 *
 * @author luis
 */
public class FecheiroTeste {
    private ArrayList<TreeSet<Integer>> dados = new ArrayList<
        TreeSet<Integer>>();
    private ArrayList<String> resltados = new ArrayList<String>();
    private ArrayList<String> linhas = new ArrayList<String>();

    public ArrayList<TreeSet<Integer>> getDados() {
        return dados;
    }

    public ArrayList<String> getResltados() {
        return resltados;
    }

    public ArrayList<String> getLinhas() {
        return linhas;
    }
}
```



```
        public void clear(){
            this.dados.clear();
            this.resltados.clear();
            this.linhas.clear();
        }

        public ArrayList <Integer> semEfeito(FechreiroTeste falhas){
            ArrayList <Integer> aux = new ArrayList <Integer>();

            for(int i=0; i < linhas.size(); i++){
                if(this.linhas.get(i).equals(falhas.linhas.get(i))){
                    aux.add(i+1);
                }
            }

            return aux;
        }
    }

}

package ats;

import com.itextpdf.text.DocumentException;
import java.awt.Color;
import java.awt.Component;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;
import javax.swing.JTable;
import javax.swing.table.DefaultTableCellRenderer;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.DefaultTableCellRenderer;

/**
 *
 * @author luis
 */

public class Interface extends javax.swing.JFrame{
```



```
private DadosEntrada dadosEntrda = new DadosEntrada();
private coiso dadosAvaliados;
private coiso dadosAvaliadosComFalhas;
private int file1 = 0;
private int file2 = 0;
private boolean ok = false;

public Interface() throws FileNotFoundException, IOException {
    initComponents();
    this.setTitle("Spectrum-based_Fault_Localization");
    this.setLocationRelativeTo(null);
    this.relatoriosPDF.setEnabled(false);
}

private void conTab( JTable tab , coiso c, int p){
    //Tabela
    DefaultTableModel Tmodel = new DefaultTableModel();
    Tmodel.addColumn("Teste");
    for(int i = 1; i < dadosEntrda.getMax()+1; i++){
        Tmodel.addColumn("B" + i);
    }
    Tmodel.addColumn("Erro");
    Iterator it = c.getMatriz().iterator();
    int i =1;
    while(it.hasNext()){
        ArrayList<Integer> aux = (ArrayList<Integer>) it.next();
        aux.add(0, i);
        aux.add(c.getRes().get(i-1));
        Tmodel.addRow(aux.toArray());
        i++;
    }
    Object aux[] = new Object[c.getProb().size()+1];
    Object aux2[] = c.getProb().toArray();
    aux[0]="PE";
    for(int j=1; j < (c.getProb().size()+1); j++){
        aux[j]=aux2[j-1];
    }
    Tmodel.addRow(aux);
    tab.setModel(Tmodel);
    tab.setAutoResizeMode(tab.AUTO_RESIZE_OFF);
    if(p == 1){
        this.pintar(tab);
    }
    else if( p == 2){
        this.pintar2(tab);
    }
}
```



```
}

@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">//
    GEN-BEGIN: initComponents
private void initComponents() {
    java.awt.GridBagConstraints gridBagConstraints;

    jFileChooser1 = new javax.swing.JFileChooser();
    jMenuItem2 = new javax.swing.JMenuItem();
    jMenuItem4 = new javax.swing.JMenuItem();
    jMenuItem6 = new javax.swing.JMenuItem();
    jTabbedPane1 = new javax.swing.JTabbedPane();
    jPanel1 = new javax.swing.JPanel();
    jScrollPane1 = new javax.swing.JScrollPane();
    jTable1 = new javax.swing.JTable() {
        public boolean isCellEditable(int rowIndex, int colIndex)
        {
            return false;
        }
    };
    jPanel2 = new javax.swing.JPanel();
    jScrollPane2 = new javax.swing.JScrollPane();
    jTextArea1 = new javax.swing.JTextArea();
    jPanel3 = new javax.swing.JPanel();
    jScrollPane3 = new javax.swing.JScrollPane();
    jTable2 = new javax.swing.JTable() {
        public boolean isCellEditable(int rowIndex, int colIndex)
        {
            return false;
        }
    };
    jMenuBar1 = new javax.swing.JMenuBar();
    jMenu1 = new javax.swing.JMenu();
    jMenuItem1 = new javax.swing.JMenuItem();
    jMenuItem3 = new javax.swing.JMenuItem();
    jMenuItem7 = new javax.swing.JMenuItem();
    relatoriosPDF = new javax.swing.JMenuItem();
    jMenu2 = new javax.swing.JMenu();
    jMenuItem5 = new javax.swing.JMenuItem();
    jMenuItem8 = new javax.swing.JMenuItem();

    jMenuItem2.setText("jMenuItem2");

    jMenuItem4.setText("jMenuItem4");

    jMenuItem6.setText("jMenuItem6");
```




```
setDefaultCloseOperation(javax.swing.WindowConstants.  
    EXIT_ON_CLOSE);  
getContentPane().setLayout(new java.awt.GridBagLayout());  
  
jPanel1.setLayout(new java.awt.GridBagLayout());  
  
jTable1.setModel(new javax.swing.table.DefaultTableModel(  
    new Object [][] {  
  
        },  
    new String [] {  
  
    }  
));  
jScrollPane1.setViewportViewView(jTable1);  
  
gridBagConstraints = new java.awt.GridBagConstraints();  
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;  
gridBagConstraints.weightx = 1.0;  
gridBagConstraints.weighty = 1.0;  
jPanel1.add(jScrollPane1, gridBagConstraints);  
  
jTabbedPane1.addTab("Tabela", jPanel1);  
  
jPanel2.setLayout(new java.awt.GridBagLayout());  
  
jTextArea1.setEditable(false);  
jTextArea1.setColumns(20);  
jTextArea1.setLineWrap(true);  
jTextArea1.setRows(5);  
jScrollPane2.setViewportViewView(jTextArea1);  
  
gridBagConstraints = new java.awt.GridBagConstraints();  
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;  
gridBagConstraints.weightx = 1.0;  
gridBagConstraints.weighty = 1.0;  
jPanel2.add(jScrollPane2, gridBagConstraints);  
  
jTabbedPane1.addTab("Relatorio", jPanel2);  
  
jPanel3.setLayout(new java.awt.GridBagLayout());  
  
jTable2.setModel(new javax.swing.table.DefaultTableModel(  
    new Object [][] {  
  
        },  
    new String [] {  
  
    }  
})
```



```
));
jScrollPane3.setViewportViewView(jTable2);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.weightx = 1.0;
gridBagConstraints.weighty = 1.0;
jPanel3.add(jScrollPane3, gridBagConstraints);

jTabbedPane1.addTab("Tabela _ testes _ com _ falhas", jPanel3);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 0;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.weightx = 1.0;
gridBagConstraints.weighty = 1.0;
getContentPane().add(jTabbedPane1, gridBagConstraints);

jMenu1.setText("Carregar");

jMenuItem1.setText("Resultados _ Esperados");
jMenuItem1.addActionListener(new java.awt.event.
    ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent
            evt) {
            jMenuItem1ActionPerformed(evt);
        }
    });
jMenu1.add(jMenuItem1);

jMenuItem3.setText("Resultados _ Dos _ Testes");
jMenuItem3.addActionListener(new java.awt.event.
    ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent
            evt) {
            jMenuItem3ActionPerformed(evt);
        }
    });
jMenu1.add(jMenuItem3);

jMenuItem7.setText("Resultados _ Dos _ Testes _ Com _ Falhas");
jMenuItem7.setEnabled(false);
jMenuItem7.addActionListener(new java.awt.event.
    ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent
            evt) {
            jMenuItem7ActionPerformed(evt);
        }
    })
```



```
});
jMenu1.add(jMenuItem7);

relatoriosPDF.setText("Gerar_Relatorio_PDF");
relatoriosPDF.addActionListener(new java.awt.event.
    ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent
            evt) {
            relatoriosPDFActionPerformed(evt);
        }
    });
jMenu1.add(relatoriosPDF);

jMenuBar1.add(jMenu1);

jMenu2.setText("Avaliar");
jMenu2.setEnabled(false);

jMenuItem5.setText("Avaliar");
jMenuItem5.addActionListener(new java.awt.event.
    ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent
            evt) {
            jMenuItem5ActionPerformed(evt);
        }
    });
jMenu2.add(jMenuItem5);

jMenuItem8.setText("Avaliar_Testes");
jMenuItem8.setEnabled(false);
jMenuItem8.addActionListener(new java.awt.event.
    ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent
            evt) {
            jMenuItem8ActionPerformed(evt);
        }
    });
jMenu2.add(jMenuItem8);

jMenuBar1.add(jMenu2);

setJMenuBar(jMenuBar1);

pack();
} // </editor-fold> // GEN-END: initComponents

private void jMenuItem1ActionPerformed(java.awt.event.ActionEvent
    evt) { // GEN-FIRST:event_jMenuItem1ActionPerformed
    JFileChooser fc = new JFileChooser();
```



```
int returnVal = fc.showOpenDialog(this);
if (returnVal == JFileChooser.APPROVE_OPTION) {
    try {
        dadosEntrda.lerFicheiroRes(fc.getSelectedFile().
            getAbsolutePath());
        file2=1;
    }
    catch (FileNotFoundException ex) {}
    catch (IOException ex) {}
}
if(file1 == file2 && file1 == 1){
    this.jMenu2.setEnabled(true);
    this.relatoriosPDF.setEnabled(true);
    dadosAvaliados = new coiso(dadosEntrda.matris(1),
        dadosEntrda.res(1));
}
else{
    this.jMenu2.setEnabled(false);
    this.relatoriosPDF.setEnabled(false);
}
}//GEN-LAST:event_jMenuItem1ActionPerformed

private void jMenuItem3ActionPerformed(java.awt.event.ActionEvent
    evt) //GEN-FIRST:event_jMenuItem3ActionPerformed
JFileChooser fc = new JFileChooser();
int returnVal = fc.showOpenDialog(this);
if (returnVal == JFileChooser.APPROVE_OPTION) {
    String aux =fc.getSelectedFile().getAbsolutePath();
    try {
        if (dadosEntrda.lerFicheiroTeste(aux,1)){
            file1 = 1;
            jMenuItem7.setEnabled(true);
        }
        else{
            JOptionPane.showMessageDialog(this, "Ficheiro _
                Invalido.", "Erro", JOptionPane.ERROR_MESSAGE)
                ;
        }
    }
    catch (FileNotFoundException ex) {}
    catch (IOException ex) {}
}
if(file1 == file2 && file1 == 1){
    this.jMenu2.setEnabled(true);
    this.relatoriosPDF.setEnabled(true);
    dadosAvaliados = new coiso(dadosEntrda.matris(1),
        dadosEntrda.res(1));
}
else{
```



```
        this.jMenu2.setEnabled(false);
        this.relatoriosPDF.setEnabled(false);
    }
} //GEN-LAST:event_jMenuItem3ActionPerformed

private void jMenuItem5ActionPerformed(java.awt.event.ActionEvent
    evt) { //GEN-FIRST:event_jMenuItem5ActionPerformed
    if(dadosEntrda.tamanho(1)) {
        if(this.dadosAvaliadosComFalhas != null){
            this.jMenuItem8.setEnabled(true);
        }
        this.ok= true;
        this.conTab(this.jTable1 , dadosAvaliados ,1);
        this.relatorio();
        this.file1 = 0;
        this.file2 = 0;
    }
    else {
        JOptionPane.showMessageDialog(this, "Os ficheiros _nao_tem
            _a_mesma_cardinalidade.", "Erro", JOptionPane.
                ERROR_MESSAGE);
    }
} //GEN-LAST:event_jMenuItem5ActionPerformed

private void relatoriosPDFActionPerformed(java.awt.event.
    ActionEvent evt) { //GEN-FIRST:
    event_relatoriosPDFActionPerformed
        JFileChooser fc = new JFileChooser();
        int returnVal = fc.showSaveDialog(this);
        if (returnVal == JFileChooser.APPROVE_OPTION) {
            try {
                Pdf relatorio = new Pdf(dadosAvaliados ,
                    dadosAvaliadosComFalhas , dadosEntrda , dadosEntrda.
                        getMax());
                String cam = fc.getSelectedFile().getAbsolutePath();
                System.out.println(cam);
                if(!fc.getSelectedFile().getName().contains(".pdf")){
                    cam = cam + ".pdf";
                }
                relatorio.gerarRelatorio(cam);
            } catch (FileNotFoundException | DocumentException ex) {}
        }
    }
} //GEN-LAST:event_relatoriosPDFActionPerformed

private void jMenuItem7ActionPerformed(java.awt.event.ActionEvent
    evt) { //GEN-FIRST:event_jMenuItem7ActionPerformed
    JFileChooser fc = new JFileChooser();
    int returnVal = fc.showOpenDialog(this);
    if (returnVal == JFileChooser.APPROVE_OPTION) {
```



```
String aux = fc.getSelectedFile().getAbsolutePath();
try {
    if (!(dadosEntrda.lerFecheiroTeste(aux, 2))) {
        JOptionPane.showMessageDialog(this, "Ficheiro _
        Invalido.", "Erro", JOptionPane.ERROR_MESSAGE)
        ;
    }
}
catch (FileNotFoundException ex) {}
catch (IOException ex) {}

dadosAvaliadosComFalhas = new coiso(dadosEntrda.matris(2)
, dadosEntrda.res(2));
if(ok){
    this.jMenuItem8.setEnabled(true);
}
}
} //GEN-LAST:event_jMenuItem7ActionPerformed

private void jMenuItem8ActionPerformed(java.awt.event.ActionEvent
    evt) { //GEN-FIRST:event_jMenuItem8ActionPerformed
    if(dadosEntrda.tamanho(2)) {
        this.conTab(this.jTable2, dadosAvaliadosComFalhas, 2);
        this.relatorio();
        this.relatoriofalhas();
        this.file1 = 0;
        this.file2 = 0;
    }
    else {
        JOptionPane.showMessageDialog(this, "Os _ ficheiros _ nao _ tem
        _ a _ mesma _ cardinalidade.", "Erro", JOptionPane.
        ERROR_MESSAGE);
    }
} //GEN-LAST:event_jMenuItem8ActionPerformed

// Variables declaration - do not modify //GEN-BEGIN:variables
private javax.swing.JFileChooser jFileChooser1;
private javax.swing.JMenu jMenuItem1;
private javax.swing.JMenu jMenuItem2;
private javax.swing.JMenuBar jMenuItemBar1;
private javax.swing.JMenuItem jMenuItem1;
private javax.swing.JMenuItem jMenuItem2;
private javax.swing.JMenuItem jMenuItem3;
private javax.swing.JMenuItem jMenuItem4;
private javax.swing.JMenuItem jMenuItem5;
private javax.swing.JMenuItem jMenuItem6;
private javax.swing.JMenuItem jMenuItem7;
private javax.swing.JMenuItem jMenuItem8;
```



```
private javax.swing.JPanel jPanel1;
private javax.swing.JPanel jPanel2;
private javax.swing.JPanel jPanel3;
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JScrollPane jScrollPane2;
private javax.swing.JScrollPane jScrollPane3;
private javax.swing.JTabbedPane jTabbedPane1;
private javax.swing.JTable jTable1;
private javax.swing.JTable jTable2;
private javax.swing.JTextArea jTextArea1;
private javax.swing.JMenuItem relatoriosPDF;
// End of variables declaration//GEN-END:variables

private void pintar(JTable tab) {
    final int rowCount = tab.getRowCount() - 1;
    final int columnCount = tab.getColumnCount() - 1;
    DefaultTableCellRenderer tableRenderer;
    tableRenderer = new DefaultTableCellRenderer() {
        public Component getTableCellRendererComponent(JTable
            table, Object value,
                boolean isSelected, boolean
                    hasFocus, int row, int column)
        {
            super.getTableCellRendererComponent(table, value,
                isSelected,
                    hasFocus, row, column);
            if (column > 0 && row == rowCount && column <
                columnCount) {

                float valor = (Float) table.getModel().getValueAt
                    (row, column);
                if (valor > 0){
                    int red = Math.round( ( 255 - 155*valor)) ;
                    setBackground(new Color(red, 10, 10));
                }
                else
                    setBackground(null);

            } else {
                setBackground(null);
            }
            return this;
        }
    };
    tab.setDefaultRenderer(Object.class, tableRenderer);
}
```



```
}

private void pintar2(JTable tab) {
    final int rowCount = tab.getRowCount() -1 ;
    final int columnCount = tab.getColumnCount() -1 ;
    final ArrayList<Integer> semEfeito = this.dadosEntrda.
        testCoverage();
    DefaultTableCellRenderer tableRenderer;
    tableRenderer = new DefaultTableCellRenderer() {
        public Component getTableCellRendererComponent(JTable
            table, Object value,
                boolean isSelected, boolean
                    hasFocus, int row, int column)
        {
            super.getTableCellRendererComponent(table, value,
                isSelected,
                    hasFocus, row, column);
            if (column >0 && row == rowCount && column <
                columnCount) {
                float valor = (Float) table.getModel().getValueAt
                    (row, column);
                if ( valor > 0){
                    int red =Math.round( ( 255 - 155*valor)) ;
                    setBackground(new Color(red, 10, 10));
                }
                else
                    setBackground(null);

            } else {
                setBackground(null);
            }
            if (semEfeito.contains(row+1)){
                setBackground(Color.yellow);
            }
            return this;
        }
    };
    tab.setDefaultRenderer(Object.class, tableRenderer);

}

private void relatorio() {
    StringBuilder s = new StringBuilder();
    ArrayList<Bc> bc = dadosEntrda.bRanking();
    ArrayList<Integer> nubc = dadosEntrda.blocosNutilizados();
```




```
        Iterator it1 = nubc.iterator();
        s.append("Blocos_de_codigo_nao_utilizados:");
        while (it1.hasNext())
        {
            s.append(it1.next());
            if(it1.hasNext()){
                s.append(",");
            }
        }
        s.append("\n\n");

        it1 = bc.iterator();

        int i = 0;
        while (it1.hasNext() && i < 10)
        {
            s.append(it1.next().toString());
            i++;
        }
        this.jTextArea1.setText(s.toString());
    }

    private void relatoriofalhas() {
        StringBuilder s = new StringBuilder();
        ArrayList<Integer> semEfeito = this.dadosEntrda.testCoverage()
            ;

        Iterator it1 = semEfeito.iterator();
        if (!it1.hasNext()){
            s.append("Todos_os_teste_demonstram_anomalias_face_ao_comportamento_normal.\n");
        }
        else{
            s.append("Teste_que_nao_sofreram_alteracoes_com_a_injecao_de_falhas:");
            while (it1.hasNext())
            {
                s.append(it1.next());
                if(it1.hasNext()){
                    s.append(",");
                }
            }
            s.append("\n\n");
        }
        this.jTextArea1.append(s.toString());
    }
}

/*
```



```
* To change this template, choose Tools | Templates
* and open the template in the editor.
*/
package ats;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import javax.swing.JOptionPane;

/**
 *
 * @author luis
 */
public class Main {

    private DadosEntrada dadosEntrda = new DadosEntrada();
    private coiso dadosAvaliados;

    // nao impelmetado
    private void conPro(coiso c){
        //Tabela
        StringBuilder s = new StringBuilder();
        Iterator it1 = c.getProb().iterator();
        int i=0;
        while (it1.hasNext()){
            Float next = (Float) it1.next();
            if(next > 0.0F){
                if(i>0){
                    s.append(",");
                }
                s.append("(");
                s.append(i+1);
                s.append(",");
                s.append(it1.next());
                s.append(")");
            }
            i++;
        }
        s.append("\n");

        System.out.println(s.toString());
    }

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
```



```
private void relatorio1() {
    StringBuilder s = new StringBuilder();
    ArrayList<Bc> bc = dadosEntrda.bRanking();
    ArrayList<Integer> nubc = dadosEntrda.blocosNutilizados();

    // Bc not used
    Iterator it1 = nubc.iterator();
    while (it1.hasNext()){
        s.append(it1.next());
        if(it1.hasNext()){
            s.append(",");
        }
    }
    s.append("\n\n");
    System.out.println(s.toString());
}

private void relatorio2() {
    StringBuilder s = new StringBuilder();
    ArrayList<Bc> bc = dadosEntrda.bRanking();
    ArrayList<Integer> nubc = dadosEntrda.blocosNutilizados();

    // mais utilizados
    Iterator it1 = nubc.iterator();
    it1 = bc.iterator();
    int i = 0;
    while (it1.hasNext() && i < 10)
    {
        s.append(((Bc)it1.next()).getBoco());
        s.append(",");
        s.append(((Bc)it1.next()).getVezes());
        s.append("\n");
        i++;
    }
    System.out.println(s.toString());
}

private void relatoriofalhas() {
    StringBuilder s = new StringBuilder();
    ArrayList<Integer> semEfeito = this.dadosEntrda.testCoverage()
        ;

    Iterator it1 = semEfeito.iterator();
    if (!it1.hasNext()){
        s.append("0\n");
    }
    else{
        while (it1.hasNext())
```



```
        {
            s.append(it1.next());
            if(it1.hasNext()){
                s.append(", ");
            }
        }
        s.append("\n");
    }
    System.out.println(s.toString());
}

public static void main(String args[]) throws IOException {
    /* Set the Nimbus look and feel */
    //<editor-fold defaultstate="collapsed" desc=" Look and feel
        setting code (optional) ">
    /* If Nimbus (introduced in Java SE 6) is not available, stay
        with the default look and feel.
        * For details see http://download.oracle.com/javase/tutorial
        /uiswing/lookandfeel/plaf.html
        */
    try {
        for (javax.swing.UIManager.LookAndFeelInfo info : javax.
            swing.UIManager.getInstalledLookAndFeels()) {
            if ("Nimbus".equals(info.getName())) {
                javax.swing.UIManager.setLookAndFeel(info.
                    getClassName());
                break;
            }
        }
    } catch (ClassNotFoundException ex) {
        java.util.logging.Logger.getLogger(Interface.class.
            getName()).log(java.util.logging.Level.SEVERE, null,
            ex);
    } catch (InstantiationException ex) {
        java.util.logging.Logger.getLogger(Interface.class.
            getName()).log(java.util.logging.Level.SEVERE, null,
            ex);
    } catch (IllegalAccessException ex) {
        java.util.logging.Logger.getLogger(Interface.class.
            getName()).log(java.util.logging.Level.SEVERE, null,
            ex);
    } catch (javax.swing.UnsupportedLookAndFeelException ex) {
        java.util.logging.Logger.getLogger(Interface.class.
            getName()).log(java.util.logging.Level.SEVERE, null,
            ex);
    }
}
//</editor-fold>

/* Create and display the form */
```



```
Main m = new Main();
if( args.length == 0){
    Interface i = new Interface();
    i.setVisible(true);
}
else{
    try {
        int tp = Integer.parseInt( args[0] );
        if( tp > 5 && tp == 1 && tp == 2){
            System.out.println("Numero_de_parametros_
                                incorretos");
        }
        else if( args.length > 2 && args.length < 5){
            try {
                if ( !m.dadosEntrda.lerFecheiroTeste( args[1] ,
                    1) ) {
                    System.out.println(" Ficheiro_Invalido:_ " +
                        args[1]+ "._Ou_nao_foi_encontrado" );
                }
            }
            else{
                m.dadosEntrda.lerFecheiroRes( args[2] );
                if( m.dadosEntrda.tamanho(1) ) {
                    m.dadosAvaliados = new coiso( m.
                        dadosEntrda.matris(1) , m.
                        dadosEntrda.res(1) );
                    if( tp == 1 || tp == 0){
                        m.conPro( m.dadosAvaliados );
                    }
                    if( tp == 2 || tp == 0){
                        m.relatorio1();
                    }
                    if( tp == 3 || tp == 0){
                        m.relatorio2();
                    }
                }
            }
            else{
                System.out.println("Os_ficheiros ,_ "+
                    args[1]+ "_e_" + args[2] + "_nao_
                    tem_a_mesma_cardinalidade." );
            }
        }
    }
    if( args.length == 4){
        if ( !m.dadosEntrda.lerFecheiroTeste( args
            [3] , 2) ) {
            System.out.println(" Ficheiro_Invalido :
                _" +args[3]+ "._Ou_nao_foi_
                encontrado" );
        }
    }
    else{
```

```

        if(m.dadosEntrda.tamanho(2)) {
            if(tp == 4 || tp == 0){
                m.relatoriofalhas();
            }
        }
    else {
        System.out.println("Os ficheiros ,
                            "+args[1]+ " e " + args[2] +
                            "nao tem a mesma cardinalidade
                            ." );
    }
}
}
}
}
catch (FileNotFoundException ex) {}
catch (IOException ex) {}
}
else{
    System.out.println("Parametro incorreto");
}
}catch(NumberFormatException nfe) {
    System.out.println("Opcao invalida");
}
}
}

}

}

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package ats;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.Iterator;
import com.itextpdf.text.Anchor;
import com.itextpdf.text.BadElementException;
import com.itextpdf.text.BaseColor;
import com.itextpdf.text.Chapter;
import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import com.itextpdf.text.Element;
import com.itextpdf.text.Font;
```



```
import com.itextpdf.text.Paragraph;
import com.itextpdf.text.Phrase;
import com.itextpdf.text.Section;
import com.itextpdf.text.pdf.PdfPCell;
import com.itextpdf.text.pdf.PdfPTable;
import com.itextpdf.text.pdf.PdfWriter;

/**
 *
 * @author luis
 */
public class Pdf {
    private coiso dadosAvaliados;
    private coiso dadosAvaliadosComFalhas;
    private DadosEntrada dadosEntrda;
    private Integer max;

    public Pdf(coiso dadosAvaliados, coiso dadosAvaliadosComFalhas,
        DadosEntrada dadosEntrda, Integer max) {
        this.dadosAvaliados = dadosAvaliados;
        this.dadosAvaliadosComFalhas = dadosAvaliadosComFalhas;
        this.max = max;
        this.dadosEntrda=dadosEntrda;
    }

    public void gerarRelatorio(String FILE) throws
        FileNotFoundException, DocumentException{
        Font catFont = new Font(Font.FontFamily.TIMES_ROMAN, 18, Font
            .BOLD);
        Font subFont = new Font(Font.FontFamily.TIMES_ROMAN, 16, Font
            .BOLD);
        Font smallBold = new Font(Font.FontFamily.TIMES_ROMAN, 12,
            Font.BOLD);
        try {
            Document doc = new Document();
            PdfWriter.getInstance(doc, new FileOutputStream(FILE));
            doc.open();
            addMetaData(doc);
            addTitlePage(doc, catFont, smallBold);
            addContent(doc, catFont, subFont);
            doc.close();
        } catch (FileNotFoundException | DocumentException e) {}
    }
}
```



```
private void addMetaData(Document document) {
    document.addTitle("Spectrum-based_Fault_Localization");
    document.addSubject("Matrizes");
    document.addAuthor("Bruno, Luis e Tiago");
    document.addCreator("Bruno, Luis e Tiago");
}

private void addTitlePage(Document document, Font catFont, Font
smallBold) throws DocumentException{
    Paragraph preface = new Paragraph();
    addEmptyLine(preface, 1);
    preface.add(new Paragraph("Spectrum-based_Fault_Localization"
        , catFont));
    addEmptyLine(preface, 4);
    preface.add(new Paragraph("Este documento demonstra a matriz
        criada pela aplicacao assim como o vector de erros.",
        smallBold));
    addEmptyLine(preface, 8);
    document.add(preface);
}

private void addContent(Document document, Font catFont, Font
subFont) throws DocumentException {
    Anchor anchor = new Anchor("Matriz", catFont);
    anchor.setName("Matriz");
    // Second parameter is the number of the chapter
    Chapter catPart = new Chapter(new Paragraph(anchor), 1);
    // Add a table
    createTable(catPart, dadosAvaliados);
    // Now add all this to the document
    document.add(catPart);

    anchor = new Anchor("Relatorio_Blocos_de_Codigo", catFont);
    anchor.setName("Relatorio_Blocos_de_Codigo");
    // Second parameter is the number of the chapter
    catPart = new Chapter(new Paragraph(anchor), 2);
    relatorio(catPart);
    // Now add all this to the document
    document.add(catPart);
    if(this.dadosAvaliadosComFalhas != null){

        anchor = new Anchor("Relatorio_Testes", catFont);
        anchor.setName("Relatorio_Testes");
        // Second parameter is the number of the chapter
        catPart = new Chapter(new Paragraph(anchor), 3);
        relatoriofalhas(catPart);
        catPart.add(new Paragraph(" "));
        catPart.add(new Paragraph("Matriz_dos_testes_com_falhas."));
    }
}
```




```
catPart.add(new Paragraph(" "));
createTable(catPart, dadosAvaliadosComFalhas);
document.add(catPart);
}

}

private void createTable(Chapter CatPart, coiso c) throws
BadElementException {
    int colunas = this.max + 2; //2 corresponde ao teste e ao erro
    int cont = 0, nr_tab = colunas/7, resto = colunas%7, i = 0;

    for (; cont < nr_tab; cont++) { //vai fazer "nr_tab" tabelas
        if (cont == 0) { //Na primeira tabela tem que meter a
            coluna teste
            int lim = 6*(cont+1);
            PdfPTable table = new PdfPTable(7);
            //colunas
            table.addCell("Teste");
            for (; i < lim; i++) {
                table.addCell("B" + (i+1));
            }
            //linhas
            int j = 1;
            Iterator it = c.getMatriz().iterator();
            while (it.hasNext()) {
                table.addCell(" "+j);
                ArrayList<Integer> aux = (ArrayList<Integer>) it.
                    next();
                for (i = 0; i < lim; i++) {
                    table.addCell(" " + aux.get(i));
                }
                j++;
            }
            //adicionar probabilidades
            table.addCell("PE");
            for (i = 0; i < lim; i++) {
                table.addCell(" " + c.getProb().get(i));
            }
            //adicionar tabela
            CatPart.add(table);
            CatPart.newPage();
        }
        else {
            if (resto != 0) { //Garantir que o erro nao aparece aqui
                int aux_nr = i; int lim = 7*(cont+1)-1;
                PdfPTable table = new PdfPTable(7);
```



```
//colunas
for (; i < lim; i++){
    table.addCell("B" + (i+1));
}
//linhas
for(ArrayList<Integer> aux: c.getMatriz()){
    for(i = aux_nr; i < lim; i++){
        table.addCell("" + aux.get(i));
    }
}
//adicionar probabilidades
for(i = aux_nr; i < lim; i++){
    table.addCell("" + c.getProb().get(i));
}
//adicionar tabela
CatPart.add(table);
CatPart.newPage();
}
else{//Erro na ultima coluna; implica que resto e' 0
    if((cont+1) == nr_tab){
        int aux_nr = i; int lim = 7*(cont+1)-2;
        PdfPTable table = new PdfPTable(7);
        //colunas
        for (; i < lim; i++){
            table.addCell("B" + (i+1));
        }
        table.addCell("Erro");
        //linhas
        int linh = 0;
        for(ArrayList<Integer> aux: c.getMatriz()){
            for(i = aux_nr; i < lim; i++){
                table.addCell("" + aux.get(i));
            }
            //adicionar erro
            table.addCell("" + c.getRes().get(linh));
            linh++;
        }
        //adicionar probabilidades
        for(i = aux_nr; i < lim; i++){
            table.addCell("" + c.getProb().get(i));
        }
        table.addCell("");
        //adicionar tabela
        CatPart.add(table);
        CatPart.newPage();
    }
    else{ //Ainda nao e' a ultima tabela
        int aux_nr = i; int lim = 7*(cont+1)-1;
        PdfPTable table = new PdfPTable(7);
```



```
        //colunas
        for (; i < lim; i++){
            table.addCell("B" + (i+1));
        }
        //linhas
        for (ArrayList<Integer> aux: c.getMatriz()){
            for (i = aux_nr; i < lim; i++){
                table.addCell("" + aux.get(i));
            }
        }
        //adicionar probabilidades
        for (i = aux_nr; i < lim; i++){
            table.addCell("" + c.getProb().get(i));
        }
        //adicionar tabela
        CatPart.add(table);
        CatPart.newPage();
    }
}

if(resto>0){ //Se ainda sobrar colunas mas que nao chegam a 7
    //colunas
    int aux_nr = i;
    PdfPTable table = new PdfPTable(resto);
    //colunas
    for (; i < (colunas-2); i++){
        table.addCell("B" + (i+1));
    }
    table.addCell("Erro");
    //linhas
    int linh = 0;
    for (ArrayList<Integer> aux: c.getMatriz()){
        for (i = aux_nr; i < (colunas-2); i++){
            table.addCell("" + aux.get(i));
        }
        //adicionar erro
        table.addCell("" + c.getRes().get(linh));
        linh++;
    }
    //adicionar probabilidades
    for (i = aux_nr; i < (colunas-2); i++){
        table.addCell("" + c.getProb().get(i));
    }
    table.addCell("");
    //adicionar tabela
    CatPart.add(table);
    /*PdfPTable table = new PdfPTable(resto);
    int lim = res.size(), aux_nr = i;
```



```
        for(int z = 0; z < resto-1; z++){
            table.addCell("B" + (i+1));
            //linhas
            for(ArrayList<Integer> aux: matriz){
                for(i = aux_nr; i < (colunas-2); i++){
                    table.addCell(" " + aux.get(i));
                }
            }
            //adicionar probabilidades
            for(i = aux_nr; i < (colunas-2); i++){
                table.addCell(" " + prob.get(i));
            }
        }
        //Ultima coluna de erros
        table.addCell("Erro");
        i = 0;
        while(i<lim){
            table.addCell(" " + res.get(i));
            i++;
        }
        CatPart.add(table);*/
    }
}

private void relatorio(Chapter catPart) throws DocumentException
{
    ArrayList<Bc> bc = dadosEntrda.bRanking();
    ArrayList<Integer> nubc = dadosEntrda.blocosNutilizados();
    StringBuilder s = new StringBuilder();
    Font catFont = new Font(Font.FontFamily.TIMES_ROMAN, 12);

    Iterator it1 = nubc.iterator();
    catPart.add(new Paragraph("", catFont));
    s.append("Blocos de código não utilizados: ");
    while (it1.hasNext())
    {
        s.append(it1.next());
        if(it1.hasNext()){
            s.append(", ");
        }
    }
    s.append(".");
    catPart.add(new Paragraph(s.toString(), catFont));
    it1 = bc.iterator();
    while (it1.hasNext())
    {
        catPart.add(new Paragraph(it1.next().toString(), catFont));
    }
}
```



```
    }  
}  
  
private void relatoriofalhas(Chapter catPartt) throws  
    DocumentException {  
    StringBuilder s = new StringBuilder();  
    Font catFont = new Font(Font.FontFamily.TIMES_ROMAN, 12);  
    ArrayList<Integer> semFeito = this.dadosEntrda.testCoverage();  
    Iterator it1 = semFeito.iterator();  
    if (!it1.hasNext()){  
        catPartt.add(new Paragraph("Todos os teste demonstram  
            anomalias face ao comportamento normal.\n", catFont));  
    }  
    else{  
        s.append("Teste que nao sofreram alteracoes com a  
            injecao de falhas:\n");  
        while (it1.hasNext()){  
            s.append(it1.next());  
            if (it1.hasNext()){  
                s.append(", ");  
            }  
        }  
        catPartt.add(new Paragraph(s.toString(), catFont));  
    }  
}  
  
private static void addEmptyLine(Paragraph paragraph, int number)  
    {  
        for (int i = 0; i < number; i++) {  
            paragraph.add(new Paragraph(" "));  
        }  
    }  
}
```