Teste de Cobertura para C - -

Grupo 3
A67673 - André Geraldes
A61071 - Pedro Duarte
A67709 - Sandra Ferreira

2 de Dezembro de 2015

Resumo

Hoje em dia é crucial e totalmente indispensável testarmos o software que produzimos, pois, como humanos que somos, cometemos erros. Como tal, realizamos vários casos de teste com diferentes inputs sobre os nossos programas para podermos tirar conclusões sobre a sua qualidade. Mais uma vez, pondo a condição humana em causa, idealizamos os casos de teste com inputs que devolvam os outputs esperados o que nada prova sobre a correcção do software. É, portanto, necessário analisar a cobertura dos testes que fazemos correr sobre os programas.

1 Introdução

No âmbito da unidade curricular de Análise e Teste de Software, foi-nos dada a escolha de um dos vários projetos apresentados pelos docentes. O projeto que escolhemos desenvolver foi o de Testes de Cobertura para a linguagem C–.
Este projeto tem como objetivo desenvolver uma análise de cobertura de casos de teste para C–. A ideia consiste em estender o processador de C– de modo a que sempre que um conjunto de testes for executado seja possível analisar a sua cobertura. O resultado desta análise será a geração de um relatório que indica a cobertura desses casos de teste: por exemplo, indica se todas (ou que percentagem) as funções C– foram executadas/testadas, se todos os ramos das condições *if-then-else* foram testados, se todas as componentes de uma expressão lógica da condição de paragem de um ciclo foram testada,ou mesmo, se todos os blocos de código foram testados. O analisador de cobertura deve transformar/instrumentar o código C–, do programa a ser testado, com os casos de teste, de modo a este produzir informação sobre que funções/blocos de código/expressões lógicas foram usadas na execução do programa. Posteriormente, essa informação é usada para identificar as partes do código fonte que foram ou não testadas.

2 Primeira Fase de Desenvolvimento do Projeto

Para a primeira etapa de desenvolvimento do projeto começamos por fazer vários programas em C–. Cada um destes programas está anotado com instruções de

print para nos ser possível observar os caminhos que o programa toma consoante os vários *inputs* que lhe são atribuídos. Para ser possível analisar o *tracing*, distinguimos três partes distintas num programa: as instruções, as condições e os blocos básicos(sequência consecutiva de instruções com uma única entrada e uma única saída). Assim sendo, para cada programa C–, encontramos as linhas de código seguidas de um "*print*('p');*print*(num);"onde p pode corresponde aos caracteres: i-instrução; c- condição; b- bloco ou w- *while*. O num, por sua vez, representa o número sequencial de cada uma destas partes do programa. No final dos ficheiros C - - encontram-se três *prints* que indicam o número total de instruções, blocos básicos, *if/else* e *while/for*.
O *output* gerado por cada programa criado é direcionado para o ficheiro res.txt. Foi criado um *parser* com a finalidade de analisar este ficheiro e posteriormente apresentar as respetivas estatísticas. Estas estatísticas indicam quais foram as partes do código por onde o programa passou e a percentagem de instruções, condições blocos e ciclos que foram executados e ainda, no caso dos ciclos, indica ainda o número de vezes que este é executado.

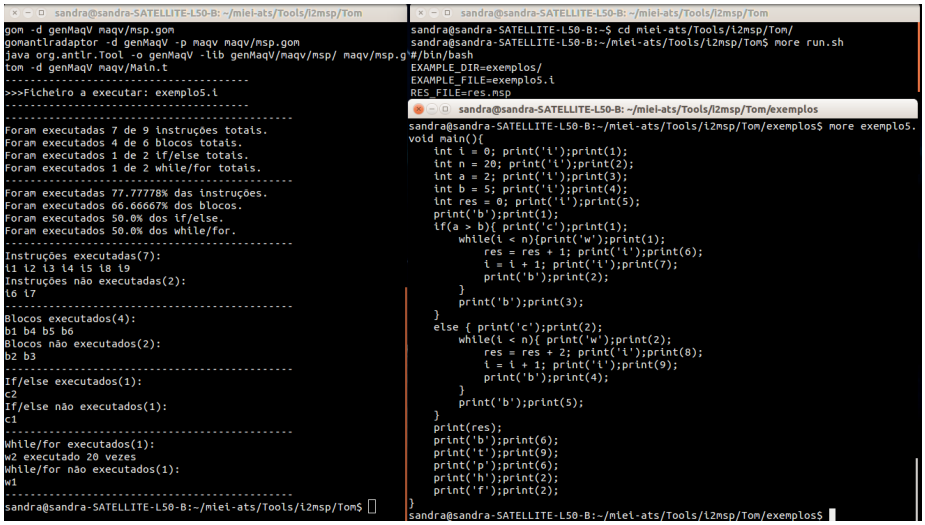

Figura 1: Estatísticas geradas pelo *parser* na análise do exemplo5.i

2.1 Conclusões Fase de Desenvolvimento

Para a primeira etapa comprometemo-nos a fazer o tracing dos programas de modo a ser possível averiguar os caminhos que os programas tomam consoante os diversos *inputs* que lhes são atribuídos. Adicionalmente, nesta fase teríamos também de gerar estatísticas para saber que percentagem de código é executada. Os prints que permitem verificar o *tracing* que é feito por um programa não são gerados automaticamente, isto é, para esta fase os prints foram lá colocados por nós. Esta foi a única coisa a que nos comprometemos e não cumprimos. A razão deste incumprimento deveu-se ao facto de no início não termos conseguido interpretar de forma correta o que se pedia no enunciado e por conseguinte não

conseguimos dividir de forma adequada o trabalhos pelas duas fases do trabalho.

3 Trabalho Futuro

Para a segunda fase de desenvolvimento do trabalho a primeira coisa a fazer é automatizar

4 Anexos

Exemplo1.i

```
void main() {
    int a;
    int b = 5; print('i');print(1);
    int r;
    a = 10; print('i');print(2);
    r = mult(a,b); print('i');print(3);
    print(r);
    print('b'); print(1);

    print('t');print(4);
    print('p');print(2);
    print('h');print(0);
}

int mult(int a, int b){
    int r;
    r = a * b; print('i');print(4);
    print('b'); print(2);
    return r;
}
```

Exemplo2.i

```
void main() {
    int i;
    i = 0; print('i');print(1);
    int a = 1; print('i');print(2);
    int b = 4; print('i');print(3);
    int c = 2; print('i');print(4);

    print('b');print(1);
    if(a > b){ print('c');print(1);
        b = 5; print('i');print(5);
        print('b');print(2);
    }
    else { print('c');print(2);
        b = 3; print('i');print(6);
        print('b');print(3);
```

```
    }

    c = a + b; print('i');print(7);
    print('b');print(4);
    /*  t -> total de instruçoes */
    /*  p -> total de blocos */
    /*  h -> total de if elses */
    print('t');print(7);
    print('p');print(4);
    print('h');print(2);
}
```

Exemplo3.i

```
void main () {
int a; a = 3; print('i');print(1);
int b; b = 5; print('i');print(2);
int c; c = 15; print('i');print(3);
int res; res=maxTres(a,b,c); print('i');print(4);
print(res);print('i');print(5);

print('b');print(1);

print('t');print(10);
    print('p');print(6);
    print('h');print(4);
}

int maxTres (int x, int y, int z){
int max;

if(x>=y){
if (x>=z){print('c');print(1);
        max=x;print('i');print(6);

print('b');print(2);
}
else{print('c');print(2);
    max=z;print('i');print(7);

print('b');print(3);
}
}
else{print('c');print(3);
if(y>=z){print('c');print(3);
max=y;print('i');print(8);

print('b');print(4);
}
else{print('c');print(4);
max=z;print('i');print(9);
```

```
print('b');print(5);
}
}
return max;print('i');print(10);

print('b');print(6);
}
```

Exemplo4.i

```
void main (){
int l1=5; print('i'); print(1);
int l2=6; print('i'); print(2);
int res; res = perimetro(2,l1,l2); print('i'); print(3);
/* fig 1-> quadrado
   fig 2-> retângulo*/
print(res); print('i'); print(4);

print('b'); print(1);

print('t');print(8);
    print('p');print(5);
    print('h');print(2);
}

int perimetro(int f, int a, int b){
int p=0; print('i'); print(5);

print('b'); print(2);

if(f==1){ print('c'); print(1);
p=4*a; print('i'); print(6);

print('b'); print(3);
}
if(f==2){ print('c'); print(2);
p=2*a+2*b; print('i'); print(7);

print('b'); print(4);
}

return p;print('i'); print(8);

print('b'); print(5);
}
```