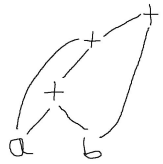
Lista 2
Peter Brendel
Guilherme Utiama

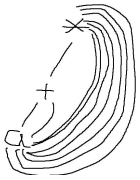
---

1 -
a) a + b + (a + b)

b) a + b + a + b



c) a + a ((a + a + a + (a + a + a + a)))



2 -

a) while(TRUE)
        if(a > 58)
            break;

L1: if not TRUE goto L2
        if a > 58 goto L2
        goto L1
L2:

b) int fat(n){
        if (n==1) return 1 ;
    return n ∗ fat(n−1);
    }

int main(int argc , char ∗ argv[ ]) {
        a = fat(5) ;
        return 0 ;
}

```
goto L1
f:
        n := p^-1
        if n==1 return 1
        n := n-1
        param n
        _a := call f, 1
        _b := _a*n
        _c := _a+_b
        return _c

L1:
        param 5
        _a := call f, 1
        return 0
```

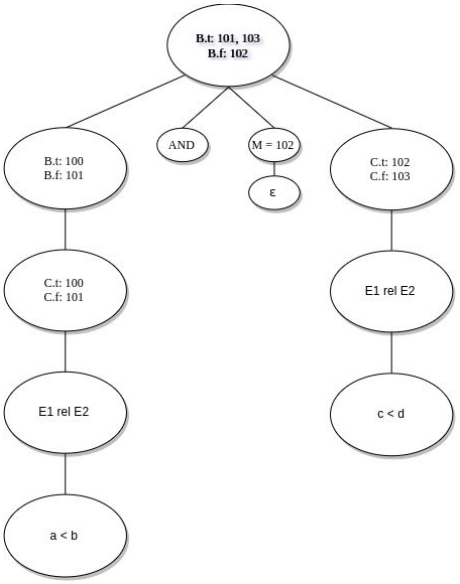
3- Código curto circuito: Quando se tem uma expressão booleana, onde não precisa-se verificar todas as respectivas respostas, pois, independente do resultado de seu anterior ou posterior, sabe-se que a sentença é verdadeira ou falsa.
Por exemplo:

```
boolean flag = true, ans = false;
        if(flag or ans){
                …;
        }
        else if(flag and ans){
                …;
         }
```

Sabemos que no primeiro if, temos a verificação de or, onde se uma das duas variáveis forem verdadeiras, ela não será dependente da outra. No segundo caso, se uma das duas forem falsas, independente da outra, ela não será aceita.

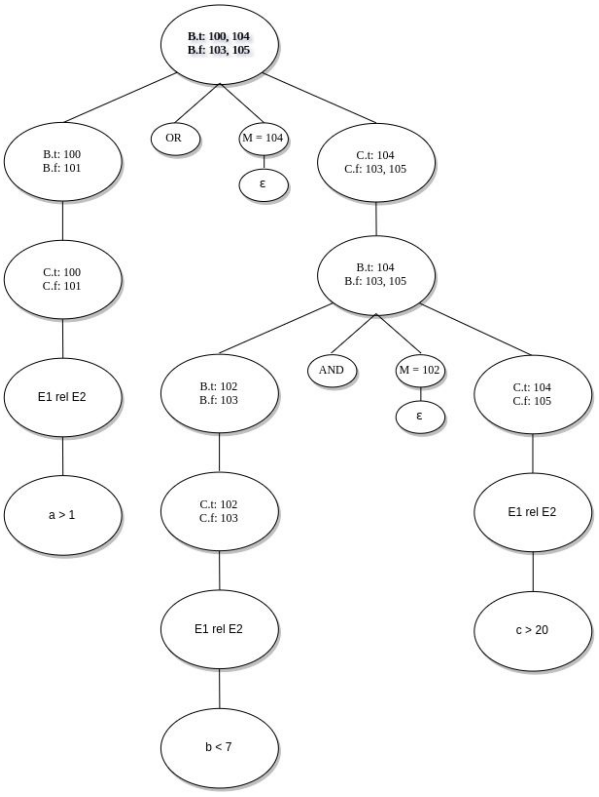
4- Utilizando a gramatica com backpatching dada em aula (slide), mostre o codigo gerado e os elementos da lista true e false que sobraram para as seguintes expressoes:

a) a < b and (c < d)



100: if a < b goto 102
101: goto __
102: if c < d goto __
103: goto__

b) a > 1 or (b < 7 and c > 20)



100: if a > 1 goto__
101: goto 102
102: if b < 7 goto 104
103: goto__
104: if c > 20 goto__

5- Primeiro o compilador deve implementar as abstrações daquela linguagem que está sendo utilizada.Após o reconhecimento da linguagem, ele cria um ambiente de gerenciamento de execução. Este ambriente lida com:

- As ligações entre funções;
- O mecanismo de passagem de parâmetros;

Entre outras funcionalidades. Com isso, ele irá separar a memória em “Tipos”, sendo eles:

- Code: Seu tamanho é determinado na compilação, nele ficam os códigos fontes.
- Static: Nele ficam armazenados variáveis globais e estáticas, por exemplo, variáveis com valores pré definidas.
- Heap: Ficam os dados alocados dinamicamente
- Stack: Armazena os registros de ativação (ex: contador de programa e registradores).

E por fim, tem a memória livre, onde, sabemos que heap e stack são partes de memórias alocados, onde seu tamanho é sempre realocado, assim, eles poderão mudar seu tamanho, e para isso, precisa-se de um local da memória livre, caso eles precisam de mais espaço.

6- A stack é subdividida em seções que são alocadas dinamicamente conforme há a necessidade de chamar novas rotinas, cada uma das seções da stack possuem suas variáveis carregadas e limitadas a seu escopo, ao fim de uma rotina o sistema operacionais desempilha a seção referente a ela e o fluxo do programa volta a ocorrer a partir do ponto onde ocorreu a chamada anterior.

7- Todas as informações armazenadas numa pilha para gerenciar uma ativação de algum dos procedimentos se chama registro de ativação ou quadro (frame).
Basicamente, quando se armazena uma variável na pilha, e uma hora quando é chamada a variável para alguma utilização, essa chamada é reconhecida como ativação de registro.