COMPILADORES

Exame final – 11/12/2009, Prof. Marcus Ramos

1. (1 ponto) Mostre, utilizando a notação dos Diagramas-T:

   - A maneira como foi feito o desenvolvimento do compilador para a linguagem U (desde o código-fonte Java até a versão executável na máquina x86);

     Java: (U → TAM)/Java em (Java → JVM)/x86 resultando em (U → TAM)/JVM
     Javac: (U → TAM)/JVM em JVM/x86

   - A maneira como é feita a compilação e a execução de programas escritos na linguagem U (supondo geração de código para máquina TAM e a existência de um interpretador para essa linguagem na máquina x86).

     P(U) em (U → TAM)/JVM em JVM/x86 resultando em P(TAM)
     P(TAM) em TAM/x86

Considere *L* a linguagem definida pela expressão regular $a^*b^*a^*$, sobre o alfabeto {*a*,*b*} e responda às questões 2 e 3.

2. (2 pontos) Obtenha:

   - Uma gramática livre de contexto que gere *L* e que <u>não seja</u> LL(1); justifique a sua resposta.

     S → XYX
     X → aX | ε
     Y → bY | ε

     Não é LL(1) pois starter (aX) = follow (X) = starter (YX) = {a}.

   - Uma gramática livre de contexto que gere *L* e que <u>seja</u> LL(1); justifique a sua resposta.

     S → aS | bX | ε
     X → bX | aY | ε
     Y → aY | ε

     É LL(1) pois starter (aS) ∩ starter (bX) = {} e starter (bX) ∩ starter (aY) = {}.

3. (2 pontos) Obtenha o esboço de um reconhecedor sintático recursivo descendente escrito em Java para a linguagem *L*.

```
void parseS(){
       if (c=='a') {takeIt(); parseS();}
             else if (c=='b') {takeIt(); parseX();}
       }
void parseX(){
       if (c=='b') {takeIt(); parseX();}
             else if (c=='a') {takeIt(); parseY();}
       }
void parseY(){
       if (c=='a') {takeIt(); parseY();}
       }
```

4. (1 ponto) Considere uma linguagem que possui tipos dinâmicos, ou seja, uma linguagem onde os tipos das variáveis podem sofrer modificação durante a execução do programa. Que estratégia deve ser usada para se fazer análise de contexto em programas escritos nessa linguagem? O que muda em relação à estrátegia estudada em sala de aula para linguagens com tipos estáticos?

No caso de linguagens com tipos dinâmicos o compilador não pode fazer a verificação de tipos durante o tempo de compilação, devendo gerar código com informações suficientes para que as mesmas ocorram durante o tempo de execução. Isso, naturalmente, onera o tamanho do programa-objeto, assim como o seu tempo de execução. É o que ocorre, por exemplo, no caso de linguagens com tipos estáticos mas cujos valores das expressões indexadoras em agregados homogêneos só podem ser verificados, de uma forma geral, durante a execução do programa.

5. (2 pontos) Obtenha um padrão de geração de código para acessar os valores armazenados nos elementos de matrizes usando máquinas de pilha. A execução das instruções geradas por esse padrão deve deixar, no topo da pilha, o valor armazenado na linha e na coluna da matriz referenciada. Considere a função de código *load* e a sintaxe *load* (`M, <exp1>, <exp2>, min1, min2, size1, size2`), onde `M` é o nome da matriz, `<exp1>` e `<exp2>` são, respectivamente, as expressões indexadoras das linhas e das colunas, `min1` e `min2` os índices iniciais das linhas e das colunas e `size1` e `size2` correspondem ao número de bytes ocupados por uma linha inteira e por uma única célula de `M`. Considere dadas as funções de código *evaluate* (`<exp>`), que gera o código que deixa no topo da pilha o valor de `<exp>`, e *base* (`M`), que deixa no topo da pilha o endereço do primeiro byte da matriz `M`. Utilize as instruções TAM na construção do seu padrão.

```
base (M)
evaluate (<exp1>)
LOADL min1
CALL sub
LOADL size1
CALL mult
CALL add
evaluate (<exp2>)
LOADL min2
CALL sub
LOADL size2
CALL mult
CALL add
LOADI
```

6. (2 pontos) Considere o seguinte trecho de programa escrito na linguagem C:

```
int x,y;
int f (int a, int b) {
    int m,n;
    ...
    }
int g (int c) {
    int p,q;
    ...
    }
int main () {
    int r,s;
    ...
    }
```

Considere o fluxo de chamadas `main` → `g` → `f` → `f`.

- Mostre a situação da pilha de execução nesse momento (com LD, LE, LB, SB, ST, variáveis e parâmetros);

```
X ← SB
Y
** main **
r
s
** g **
c
LE (vazio)
LD (main)
ER
p
```

```
q
** f **
a
b
LE (vazio)
LD (g)
ER
m
n
** f2 **
a
b
LE (vazio) ← LB
LD (f1)
ER
m
n ← ST
```

- Especifique o endereço (deslocamento+registrador) de todas as variáveis e parâmetros vísiveis durante a execução da função `f`.

```
x: 0[SB]
y: 1[SB]
a: -2[LB]
b: -1[LB]
m: 3[LB]
n: 4[LB]
```