Universidade do Porto, Faculdade de Engenharia (FEUP)

Mestrado Integrado em Engenharia Informática e de Computadores

EIC0028-2S – COMPILADORES

Teste - 2 de Maio de 2010 *Duração total (I + II): 1 hora e 30 minutos*

Nota: Na parte I as respostas erradas têm cotação negativa. Numa pergunta com 4 alternativas, uma resposta errada tem uma cotação negativa igual a 1/3 da cotação da pergunta. Numa pergunta com 2 alternativas, a resposta errada tem uma cotação negativa igual à cotação da pergunta. As perguntas não respondidas têm cotação 0.

Nome: _____ Número: _____

1. PARTE I (9 valores)

2. Análise Sintáctica

2.1 [0.75 val] O analisador sintáctico ascendente LR(0) permite:

☐ Implementar linguagens que possuam apenas recursividade à esquerda.

☒ Nenhuma das outras alíneas está correcta.

☐ Implementações software mais simples do que as necessárias para o analisador LL(1).

☐ Implementar gramáticas ambíguas.

2.2 [0.75 val] No analisador sintáctico ascendente LR(0):

☐ É sempre construída a árvore sintáctica abstracta.

☒ A árvore sintáctica concreta pode ser construída durante a análise sintáctica.

☐ A árvore sintáctica concreta só pode ser construída depois da análise sintáctica ter terminado.

☐ Nenhuma das outras alíneas está correcta.

3. Representação Intermédia de Baixo-Nível (*low-level intermediate representation*)

3.1 [0.75 val] O principal objectivo da representação intermédia de baixo nível é:

☐ Representar o programa num formato livre de ambiguidades.

☐ Identificar explicitamente as estruturas da linguagem relacionadas com *loops* e *ifs*.

☒ Fornecer uma representação do programa próxima da máquina alvo.

☐ Fornecer uma representação que facilite a análise semântica.

3.2 [0.75 val] A representação intermédia de baixo nível:

☐ Pode ser uma imagem da AST (*abstract syntax tree*) que representa o programa.

☐ Pode ser baseada em árvores de expressões em que as estruturas existentes no programa são mantidas.

☐ Pode ser uma lista de instruções no código fonte do programa a compilar.

☒ Nenhuma das outras alíneas está correcta.

4. Análise do Fluxo de Dados (*dataflow analysis*)

4.1 [0.75 val] Relativamente ao algoritmo iterativo de análise de fluxo de dados para determinar o tempo de vida das variáveis:

☐ O ponto fixo é obtido quando não houver alterações nos conjuntos das definições e dos usos.

☐ O ponto fixo é obtido quando as instruções tiverem sido todas visitadas uma vez.

☐ O ponto fixo é obtido quando todos os caminhos do grafo tiverem sido percorridos.

☒ Nenhuma das outras alíneas está correcta.

4.2 [0.75 val] A análise de fluxo de dados para determinar o tempo de vida das variáveis necessita de um algoritmo que itera até ao ponto fixo:

☐ devido ao facto de nunca se saber a ordem pela qual temos de visitar as instruções.

☒ devido ao facto da possível existência de laços.

☐ sempre que existam conjuntos de *live-in* e de *live-out* previamente determinados.

☐ sempre que não seja utilizado um grafo para a representação intermédia.

5. Alocação de Registos

5.1 [0.75 val] O grafo de interferências:

☐ pode ser obtido analisando as variáveis utilizadas por cada instrução no código.

☐ ilustra fundamentalmente as possibilidades que temos de agrupar variáveis.

☒ é construído depois de cada análise do tempo de vida das variáveis.

☐ é obtido directamente pela análise de fluxo de dados.

5.2 [0.75 val] Indique a opção correcta:

☐ Não precisamos de fazer *spilling* se considerarmos sempre que o processador tem um número muito elevado de registos disponíveis.

☐ Quando necessitamos de fazer *spilling* temos sempre de voltar a fazer a alocação de registos.

☒ Caso tenhamos de utilizar registos auxiliares para armazenar valores para as instruções de *spilling*, depois de identificarmos as variáveis que necessitam de *spilling* e de realizarmos a alocação de registos para as outras variáveis, temos de voltar a realizar a análise do tempo de vida.

☐ Podemos eliminar *spilling* quando temos variáveis com poucas interferências de tempos de vida.

5.3 [0,75 val] A utilização de *register coalescing*:

☐ é apenas uma forma de reduzir o tamanho do grafo de interferências.

☐ não traz grandes vantagem pois a própria coloração de grafos pode decidir de qualquer modo se atribui o mesmo registo para as duas variáveis ou não.

☒ pode originar dificuldades na coloração do grafo de interferências.

☐ é apenas uma forma de aumentar o número de vizinhos de um nó no grafo de interferências.

6. [2,25 val] Indique se cada uma das seguintes afirmações é verdadeira ou falsa:

V	F	
☐	☒	Se existirem conflitos na tabela do *parser* LR(0) para uma dada gramática implica sempre que a gramática é ambígua.
☒	☐	A representação intermédia de nível baixo deve reflectir a forma planar com que as variáveis são armazenadas em memória.
☐	☒	A alocação de registos é sempre um processo iterativo.
☒	☐	Na prática, não é aconselhável implementar analisadores sintácticos LR sem ser com a utilização de um gerador de *parsers* LR.
☒	☐	O conjunto das gramáticas implementáveis com o analisador sintáctico do tipo SLR(1) inclui todas as gramáticas implementáveis com o LR(0).
☒	☐	O cálculo do tempo de vida das variáveis é feito por um algoritmo iterativo cujos resultados finais não variam com a ordem com que as instruções de uma função são analisadas.
☒	☐	A selecção de instruções pode ser feita com o uso de templates (esqueletos) e de algoritmos de cobertura sobre árvores ou grafos que representam as expressões.
☐	☒	Para o *spilling* é conveniente associar o mesmo registo (ou a mesma variável) para ser usado(a) em todos os loads que tiverem de ser feitos para ir buscar o valor de uma variável à memória.
☒	☐	A implementação de uma gramática LR(0) pode ser sempre conseguida com o uso de duas pilhas e de um autómato finito determinista.
☒	☐	Se uma gramática tem uma variável A cujo Follow(A)={2,'a'} implica que no *parser* LR(0) se pode eliminar as reduções de A em todas as colunas da tabela do *parser* LR(0) que não incluam 2 e 'a'.