بسمه تعالی

آزمون پایان ترم کامپایلر

تاریخ: ۴ تیر ۱۳۹۸
مدت امتحان: ۱۸۰
مدرس: مجتهدی

۱. (۵۰نمره) گرامر زیر را در نظر بگیرید و برای آن الگوریتم SLR را شرح دهید. تمامی مراحل برای الگوریتم، شامل (۱) توضیح کلی عملکرد الگوریتم، (۲) رسم NFA برای آن، (۳) رسم DFA برای آن و (۴) تشخیص SLR بودن را توضیح دهید.

$$S \longrightarrow L = R \mid R$$
$$L \longrightarrow *R \mid \mathsf{id}$$
$$R \longrightarrow L$$

هشدار: به تمایز بین الگوریتم‌های SLR و LR_0 توجه داشته باشید.

جواب. این گرامر SLR نیست.

۲. (۱۰نمره) تعریف handle را بیان کنید.

جواب. در وضعیّت $\alpha\beta|\gamma$، گوییم β یک handle است اگر reduction $X \to \beta$ بتواند مآلاً به parse رشته بیانجامد. به عبارت دقیق‌تر داشته باشیم:

$$S \Rightarrow^* \alpha X \gamma \Rightarrow \alpha\beta\gamma$$

۳. (۲۰نمره) Operationa Semantic یک زبان برنامه‌نویسی عجیب به‌صورت زیر نوشته شده است. تشریح کنید که دستور $f(e_1, \ldots, e_s)$ دقیقا چه‌طور کار می‌کند.

$$\frac{\begin{array}{l} so, S_1, E \vdash e_1 : v_1, S_2 \\ so, S_2, E \vdash e_2 : v_2, S_3 \\ \vdots \\ so, S_n, E \vdash e_n : v_n, S_{n+1} \\ so, S_{n+1}, E \vdash e_1 : v'_1, S_{n+2} \\ v'_1 = X(a_1 = l_{a_1}, \ldots, a_m = l_{a_m}) \\ implementation(X, f) = (x_1, \ldots, x_n, e_{n+1}) \\ l_{x_i} = newloc(S_{n+2}), \text{ for } i = 1 \ldots n \text{ and each } l_{x_i} \text{ is distinct} \\ S_{n+3} = S_{n+2}[v_1/l_{x_1}, \ldots, v_n/l_{x_n}] \\ v'_1, S_{n+3}, [a_1 : l_{a_1}, \ldots, a_m : l_{a_m}, x_1 : l_{x_1}, \ldots, x_n : l_{x_n}] \vdash e_{n+1} : v_{n+1}, S_{n+4} \end{array}}{so, S_1, E \vdash f(e_1, \ldots, e_n) : v_{n+1}, S_{n+4}}$$

جواب. برای سهولت توضیح، فرض می‌کنیم $n = ۱$. ابتدا مقدار $e_۱$ را (مثلا $v_۱$) محاسبه می‌کند. سپس مقدار $e_۲$ را ($v_۲$) محاسبه می‌کند. سپس دوباره $e_۱$ را ($v'_۱$) محاسبه می‌کند. تایپ مربوط به $v'_۱$ را به‌دست می‌آورد. مثلا X. سپس تعریف تابع f را در کلاس X می‌یابد. تابع f را آن‌طور که در کلاس X تعریف شده، روی مقادیر $v_۱$ و $v_۲$ به‌عنوان پارامتر صوری اجرا می‌کنیم. هم‌چنین در حین اجرا از $v'_۱$ به‌عنوان مقدار $self$ استفاده می‌کنیم.

۴. (۱۵نمره) توضیح دهید که رنگ‌آمیزی گراف در Optimization کامپایلر چگونه در Register Allocation به‌کار می‌آید.

جواب. پاسخ از صفحه‌ی ۵۵۷ کتاب Aho:

Once the instructions have been selected, a second pass assigns physical registers to symbolic ones. The goal is to find an assignment that minimizes the cost of spills.

In the second pass, for each procedure a *register-interference graph* is constructed in which the nodes are symbolic registers and an edge connects two nodes if one is live at a point where the other is defined. For example, a register-interference graph for Fig. 8.17 would have nodes for names `a` and `d`. In block B_1, `a` is live at the second statement, which defines `d`; therefore, in the graph there would be an edge between the nodes for `a` and `d`.

An attempt is made to color the register-interference graph using k colors, where k is the number of assignable registers. A graph is said to be *colored* if each node has been assigned a color in such a way that no two adjacent nodes have the same color. A color represents a register, and the color makes sure that no two symbolic registers that can interfere with each other are assigned the same physical register.

Although the problem of determining whether a graph is k-colorable is NP-complete in general, the following heuristic technique can usually be used to do the coloring quickly in practice. Suppose a node n in a graph G has fewer than k neighbors (nodes connected to n by an edge). Remove n and its edges from G to obtain a graph G'. A k-coloring of G' can be extended to a k-coloring of G by assigning n a color not assigned to any of its neighbors.

By repeatedly eliminating nodes having fewer than k edges from the register-interference graph, either we obtain the empty graph, in which case we can produce a k-coloring for the original graph by coloring the nodes in the reverse order in which they were removed, or we obtain a graph in which each node has k or more adjacent nodes. In the latter case a k-coloring is no longer possible. At this point a node is spilled by introducing code to store and reload the register. Chaitin has devised several heuristics for choosing the node to spill. A general rule is to avoid introducing spill code into inner loops.

۵. (آ) (۱۰نمره) تعریف Basic Block را بیان کنید.

جواب. پاسخ از صفحه‌ی ۵۲۵ کتاب Aho:

Partition the intermediate code into *basic blocks*, which are maximal sequences of consecutive three-address instructions with the properties that

(a) The flow of control can only enter the basic block through the first instruction in the block. That is, there are no jumps into the middle of the block.

(b) Control will leave the block without halting or branching, except possibly at the last instruction in the block.

(ب) (۱۵نمره) فرض کنید سطرهای زیر یک Basic Block باشد طوری‌که در خروج از آن، هیچ‌یک از متغیّرها زنده نیستند. تعیین کنید که در هر موقعیت از Block کدام یک متغیرها زنده‌اند.

$$a = b + c$$
$$c = d + e$$
$$d = e + b$$
$$b = a + d$$

جواب.

- پس از سطر ۴: هیچ متغیّری زنده نیست.
- پس از سطر ۳: a, d زنده هستند.
- پس از سطر ۲: a, b, e زنده هستند.
- پس از سطر ۱: a, b, d, e زنده هستند.
- پیش از سطر۱: b, c, d, e زنده هستند.

۶. تابع زیر برای Global Register Allocation به‌کار می‌آید.

$$\sum_{\text{blocks } B \text{ in the Loop } L} \mathsf{use}(x, B) + ۲ * \mathsf{live}(x, B)$$

(آ) (۱۰نمره) تعریف تابع‌های use و live را بیان کنید.

(ب) (۱۰نمره) توضیح دهید که این تابع چگونه می‌تواند در تصمیم‌گیری برای Global Register Allocation به کار آید.

جواب. پاسخ از صفحه‌ی ۵۵۴ کتاب Aho:

On the debit side, if x is live on entry to the loop header, we must load x into its register just before entering loop L. This load costs two units. Similarly, for each exit block B of loop L at which x is live on entry to some successor of B outside of L, we must store x at a cost of two. However, on the assumption that the loop is iterated many times, we may neglect these debits since they occur only once each time we enter the loop. Thus, an approximate formula for the benefit to be realized from allocating a register x within loop L is

$$\sum_{\text{blocks } B \text{ in } L} use(x, B) + 2 * live(x, B) \tag{8.1}$$

where $use(x, B)$ is the number of times x is used in B prior to any definition of x; $live(x, B)$ is 1 if x is live on exit from B and is assigned a value in B, and $live(x, B)$ is 0 otherwise. Note that (8.1) is approximate, because not all blocks in a loop are executed with equal frequency and also because (8.1) is based on the assumption that a loop is iterated many times. On specific machines a formula analogous to (8.1), but possibly quite different from it, would have to be developed.

موفق باشید.