Exercises on Compilers

Jordi Cortadella

February 7, 2022

Code generation

1. Generate code for the following statements:

   - `a = b[i] + c[i]`
   - `a[i] = b*c - b*d`
   - `x = f(y+1) + 2`
   - `*(p+2) = *(q+1) + y`

   The code should be generated as any compiler would do by traversing the AST (i.e., no optimizations should be applied). You can assume that all basic data types are integers (4 bytes) and all the rhs expressions are evaluated before the lhs expressions.

2. Generate code for the following C++ function:

```
int search(int *a, int n, int x) {
    int i = 0;
    while (i < n and a[i] != x) i = i + 1;
    if (i < n) return i;
    return -1;
}
```

   The code should be generated as any compiler would do by traversing the AST (i.e., no optimizations should be applied).

   Note: Use backpatching for the evaluation of Boolean expressions.

3. Consider the following C++ function:

```
int f(vector<int>& A, double d) {
    struct {double a; int b;} s;
    vector<int> X(100);
    int i;
    ...
    L1:
    X.resize(200);
    ...
    while (i > 0) f
        int j = 0;
        vector<double> M(i);
        L2:
        ...
    }
    ...
    L3:
    return X[0];
}
```

- Describe the contents of the symbol table when compiling the code at lines L1, L2 and L3. For each entry, indicate the symbol, the type of symbol (local var, parameter), the data type, the size, and the offset in the stack.
- Describe the allocated memory (stack and heap) at the same lines during the execution of the program.
- Describe a situation in which a program could generate memory leaks, i.e., not all the allocated objects are deallocated before the completion of the program.
- Consider a C++ program that never uses the new/delete statements. Can it generate memory leaks? Reason your answer.

For the description of the symbol table, you can assume that `int` and `double` use 4 and 8 bytes, respectively, and that a `vector<T>` descriptor is a `struct` with three fields: *size* (`int`), *capacity* (`int`) and *data* (`T*`). You can assume that pointers use 8 bytes.

For the activation record of the function, you can assume that all parameters and variables are allocated in the stack and that the return address and the previous FP are stored in the region FP[0..15]. The result is stored in FP+16 and the parameters are stored after the result from last to first.

4. Consider the following data structure and assignment:

```
struct {
    int a;
    int b[10];                  A[i+2*j].c = A[n].b[k] + 1;
    int c;
} A[100];
```

- Describe the AST of the data structure and assignment.
- Calculate the size of the data structure and the offset associated to each field of the structure. Assume that `int` takes 4 bytes. You must annotate the information next to the nodes in the AST.
- Generate the code of the assignment. Each node of the AST must be annotated with the variable (and offset, when applicable) that contains the value calculated for that node. You must generated unoptimized code, in a similar way as a code generator would do when executing a post-order traversal of the AST.

5. De Morgan's law says that $\neg(p \vee q) \equiv \neg p \wedge \neg q$.

Show that the code generated by the following two statements (using backpatching) is identical:

- `if not (p or q) then S1 else S2`
- `if not p and not q then S1 else S2`

6. Consider the following code with nested static scopes:

```
func P (int a) {
    int x;
    func Q (int b) {
        int y;
        x = a + y;       // S1
    }
    func R () {
        int a;
        func S() {
            int c;
            c = a - x;   // S2
            P(a);        // S3
        }
    }
}
```

Generate code for the three statements in the lines with comments `S1`, `S2` and `S3`.