Compiler Construction (List 2)

Hans de Nivelle

21.10.2015

1. Consider the following, recursive implementation of **gcd**:

   ```
   size_t gcd( size_t x, size_t y )
   {
      if( x == 0 ) return y;
      if( y == 0 ) return x;

      if( x < y )
         return gcd( x, y - x );
      else
         return gcd( x - y, y );
   }
   ```

   Generate unoptimized LLVM code for this function, using **clang**. After that, rewrite the function into an LLVM function that has all local variables `x`, `y` in registers. It won't contain any **alloca** statements anymore.

   Try to resist the temptation to generate such function automatically through optimization.

2. Come up with some reasonable $O(n^2)$ sorting function, and implement in LLVM:

   ```
   void sort( unsigned int* p1, unsigned int* p2 )
   {

   }
   ```

   Again, you may use a clang generated version as starting point. After that, you can rewrite it to have all local variables in memory. As in Task 1, we assume that you stay on the narrow road.

3. Consider $F(n)$ defined by

$$\begin{cases} F(n) = n/2 & \text{if } n \text{ is even} \\ F(n) = 3n+1 & \text{if } n \text{ is odd} \end{cases}$$

It is conjectured that every number $n > 0$ has $F^i(n) = 1$, for some i. (The Collatz conjecture.) We call the smallest i with $F^i(n) = 1$ the *stopping time* of n. We want to find the number with the highest stopping time between 1 and **upperbound**:

```
size_t longestsequence( size_t upperbound )
{
   size_t longesti = 0;
   size_t longeststop = 0;
   for( size_t i = 1; i < upperbound; ++ i )
   {
      size_t length = 0;
      size_t val = i;

      while( val != 1 )
      {
         if( val & 1 )
            val = val * 3 + 1;
         else
            val = val / 2;
         ++ length;
      }
      if( length > longeststop )
      {
         longesti = i;
         longeststop = length;
      }
   }

   return longesti;
}
```

Assume that all variables are local (will be in registers).

(a) Rewrite function **longestsequence** into a flow diagram that is in SSA.

(b) After that, translate your code into LLVM. You may use an unoptimized translation as starting point.

4. Assume

```
struct tree
{
   double x;
   double y;
   tree* left;
```

```
      tree* right;
   };
```

and a function `const double* find( double x, const tree* t )` that, assuming that `x` occurs in the tree, returns a pointer to the corresponding `y` in the tree. If `x` does not occur in the tree, it should return the null pointer. You may assume that the tree is sorted in increasing order.

As above, your function should have only local variables, and you should succesfully resist the temptation to obtain this function through calling **clang** with optimization on. It is still good to use **clang** to create an unoptimized function as starting point.