CS 434 Meeting 28— 4/24/02

Announcements

1. Midterm available today through next Tuesday

Value Numbering (cont.)

1. Last time I explained a procedure for identifying subexpressions that are guaranteed to produce the same value at runtime.
   - Develop a scheme to associate a distinct number with each possibly distinct value produced during the evaluation of a series of subexpressions.
   - Store the value number for values associated with variables in their declaration descriptors.
     - Initialize each variables value number to "undefined".
     - When a variable is used for the first time, associate it with some previously unused value number (i.e. store the value number in the variable's declaration descriptor).
     - When an assignment statement is processed store the value number associated with the right hand side in the variable's descriptor.
   - Store the value numbers associated with other expression nodes in the tree in a hash table.
     - For nodes rooted at operator nodes, the hash key will be composed of the operator and the value numbers of the operands.
     - For constant nodes, the hash key will be composed of the node type (i.e. treat Nconst as an operator) and the value of the constant.
     - Don't add Ncall nodes in the hash table. Each call gets a new value number.
     - If an expression's key is found in the hash table, use the associated value number stored in the table.
     - If an expression's key is not found, add an entry for the key and the next unused value number.
2. This scheme will work fairly well for straight line code containing nothing but references to simple, local variables. In the real world, unfortunately, there are a few more complications to deal with.
   - If the target of an assignment is an array, we have to act as if any of the elements of the array may have changed (If we encounter a[j] after assigning to a[i] we may know that i and j are the same, but we can never be sure they are different).
   - If the target of an assignment is a record component, we either have to work very hard or assume any of the record variables with such a component may have changed.
3. I had these problems in mind when I had you make each refvar node point to the variable descriptor associated with the variable being referenced. In particular, that is why I had you create dummy nodes for array element variables.
   - The dummy variable associated with an array's elements will represent all the elements of all arrays of that array type. By assigning a new value number to this dummy variable, you can ensure that the value of all future references to any element of such an array will be assumed to represent a distinct value from any earlier references.
   - Record components behave similarly. If you see an assignment to a record component, you will change the value number stored in that component's declaration descriptor. This will make your value numbering algorithm treat all future reference to this component (in any record variable of the type to which the component belongs) as distinct from earlier references.

4. The handling of array elements and record components is a fine example of the *conservative* nature of the analysis algorithms one uses when performing optimization. Such algorithms do not give exact information, but they only make "safe" mistakes. In this case, the algorithm will often fail to identify pairs of expressions that actually are CSE's. The result will be that the code generated will be less efficient (but correct!). On the other hand, the alternative of sometimes accidentally concluding that two expressions are CSE's when they are not is unacceptable. It would result in the generation of incorrect (though efficient) code.

5. While we can't be sure whether array references like a[i] and a[j] are different, we can sometimes tell when they are the same.

   - For refvar nodes, the key will be a quadruple consisting of the node type (Nrefvar), the value number of the base address sub-expression, the value of the displacement field and, finally, the value number of the variable being referenced.
   - When we process an assignment, we assign new value numbers to the variable referred to by the refvar and any of its aliases. We assign the value number of the assignments right hand side to the refvar node itself.
   - Thus, we basically both test that the actual addresses referenced are the same (which can only happen if the subscripts are equivalent) and that no assignment has changed any element of the array (or any array of the same type) since the evaluation of the previous instance of this subscripted variable (by including the value number of the referenced variable).

6. BUT WAIT! It get's worse. We still need to account for assignments involving reference parameters and for the effects of calls.

7. Suppose that a global variable X is passed as a var parameter P to some procedure. If a value is assigned to X, then the value of P will change. Similarly, assigning to P will change the value of X. In such a situation, we say that X and P are aliases of one another.

8. Basically, when we process an assignment to a non-local variable or a reference parameter it is not enough to change the value number in the descriptor of the target of the assignment. In addition, we have to change the value numbers associated with all variables that might be aliases of the assignment's target.

   - In a simple compiler like ours, we must assume that any var parameter may be aliased with any non-local variable, array element, record component or other var parameter. Basically, it can be aliased with anything other than a local variable.
     - In a compiler that was willing to do some interprocedural analysis we might gather information about parameter passing that would allow us to predict aliasing relationships.
   - We must also assume that any non-local variable, array element or record component may be aliased with any var parameter.

   Since we can't be sure that an alias relations exists, we must set the value numbers of possible aliases of the target of an assignment to the "unknown" value number rather than to the value number of the right hand side.

9. Finally, if we find a call, we must:

   - change any value numbers assigned to array elements, record components, or variables declared in or above the level of the called procedures.
   - change any value numbers assigned to variables passed as var parameters to the procedure or function.