

Optimization

Tree Optimizations Involving Constants

Tree Optimizations Involving Constants

Several optimizations involving constants may be done by tree transformations:

- Compute the values of expressions involving compile-time constants
- Eliminate useless operations involving constants (adding 0, multiplying by 1)
- Propagate values from constant assignments within a basic block
- Propagate constant values across basic block boundaries.

Constant Optimization Examples

```
program
  i,j,k,l : integer;

begin
  read (k, l);
  i := 12 + 5;
  j := (77 - 14) / 7 * k;
  k := l * 1 + 20 * 2 * i;

  write ("i = ", i + 0, " j = ", j);
  write ("k = ", k - 0, " l = ", l);
end
```

Implementing Constant Optimizations

Computing the values of expressions involving compile-time constants:

- Replace the unary or binary operation subtree with a constant node containing the value of the operation

Eliminating useless operations involving constants (such as adding 0 and multiplying by 1):

- Replace the binary operation subtree with its non-constant subtree

Propagating values from constant assignments within a basic block:

- Traverse the statements for the block in execution order
- When an assignment involves a constant RHS, put the target and the value on a “known variables” list
- When any variable is accessed, check to see if it has a known value, if so, substitute the a constant node for the variable node in the AST
- Remove variables from the list when an non-constant assignment is encountered or at the end of a basic block

Propagating constant values across basic block

We don't necessarily have to throw away our known values at the end of a basic block.

Consider the things that end blocks:

- Entrances to selection and looping statement
- Exits from the same
- Calls to procedures and functions

Under what conditions can we retain known values in each case?

This approach is a special case of a more general techniques known as flow analysis.