An Inefficient Program

Tera Computer Company
2815 Eastlake Ave E
Seattle, WA 98102

December 3, 1997

1 Overview

This nonsense C/C++ program has lots of work for improvement. Assume that all locals are initialized prior to being used.

10.

```
  <f>≡
int f(int x) {
  // ...do something interesting possibly modifying global variables...
  return(0);
}
```

11.

```
  <cse>≡
void cse(void) {
  int i, j, k, m;
  i = (j * 17);
  m = (j * 17);               // common sub expression (cse)
  k = (j * 17) * f(k);        // possible cse, if f doesn´t touch j        5
  k = f(k) * (j * 17);        // possible cse, if f doesn´t touch j
}
```

12.

<algebra>≡

```
void algebra(void ) {
  int i,j,k;
  if (i > 0) {
    j = (j − k) + k;           // ok algebraic simplification
    i = j * k;
  } else {
    i = (j / k) * k;           // very dangerous simplification
  }
  k = j / k;                   // j/k is available
}
```

13.

<constants>≡

```
void constants(void ) {
  int k,l,m,n;
  l = 0;                       // constant propagation
  m = (10 + 2 * l) + (3 * 4);  // constant folding
  m = (3 + k) + (k + 4);       // questionable algebra?
}
```

14.

<dead>≡

```
void dead(void ) {
  int i,j,k,l,m;
  l = 0;
  if (l > 0) {
    l = j;                     // dead code, never executed
  } else {
    m = i;                     // value of m never used, so dead
    m = k;                     // but m isn´t needed anyway, since...
    i = m * m;                 // we could propagate k forward into this use
    m = 12;
  }
}
```

15.

```
<tailmerge>≡
void tailmerge(void ) {
  int i, k;
  if (f(k) > 0) {
    i = f(1+k);
  } else {
    i = f(2+k);
  }
}
```

16.

```
<hiddenarith>≡
void hiddenarith(void ) {
  int i, j, k;
  int B[10], D[10];
  D[1] = f(k);               // constant fold to simplify access
  for (j = 0; j < 10; j++) {
    B[j] = i * i;            // i*i is loop invariant
                             // some parts of B[j] are loop invariant
  }
}
```

17.

```
<loopmanipulate>≡
void loopmanipulate(void ) {
  int i, j, k;
  int B[10], C[10], D[10];
  for (i = 0; i < 10; i++) { B[i] = 0; }          // loop jam
  for (i = 0; i < 10; i++) { D[i] = i; }          // loop jam
  for (i = 0; i < 10; i++) { C[i] = 10-i; }       // loop jam

  for (j = 0; j < 2; j++) {
    B[j] = f(k);             // loop unroll
  }
}
```

18.

```
  <inductionvar>≡
void inductionvar(void ) {
  int i, j, k;
  int A[10][10];
  j = f(k);
  for (i = 0; i < 10; i++) {
    A[i][j] = f(k);
  }
}
```

19.

```
  <peephole>≡
void peephole(void ) {
  int i,j,k,l;
  i = i + 1;                // use increment instruction
  i = 12 * j;               // convert to shifts and adds
  i = j + j; k = l + l; // do in parallel
}
```

20.

```
  <alias>≡
int global;
void alias(int &x)      // x is a C++ reference parameter
{
  int w, z;
  w = x * x;
  global = x * x;
  z = x * x;                // x*x is a cse only if x is not an alias for global
}
```

21.

<dec>≡

```
int dec(int x) {
  if (x > 0) {
    return (x − 1);
  } else {
    return (x);
  }
}
```

22.

<inlinedec>≡

```
void inlinedec(void ) {
  int y;
  y = dec(17);            // inline expand dec, then dead code eliminate
}
```

23.

<fact>≡

```
int fact(int x) {
  if (x <= 0) {
    return (1);
  } else {
    return (x * fact(x−1));      // tail recursion
  }
  // memoize (value cache) the function (its a pure function)
}
```