Loop-dead optimization

Sachin Shaw and Pawan Kumar *
Department of Mathematics
Indian Institute of Technology
Kharagpur-721302, India
*Corresponding author. Tel. - +91-3222-283660.
E-mail: sachinshaw@yahoo.co.in and pawan@maths.iitkgp.ernet.in

ABSTRACT

A loop is a critical part of any program. A loop is normally executed a large number of times, hence if any statement can be moved outside the loop then computation can speed-up. Loop invariant computations have been studied in great detail [1, 2]. Here a computation, which does not change inside the loop (on successive execution of the loop), is done before the loop. Such a computation involves only the loop invariant variables. These are variables, whose value does not change inside the loop.

Here we talk about a variable, which changes inside the loop, but only the last value is useful. The computation needed in the definition of this variable can be moved outside the loop (after the loop).

Formally a loop-dead variable is that which is defined in the loop but not used in the loop. Later we extend this definition as follows: A loop-dead variable is that which is not used inside the loop except in the definition of the other loop dead variables.

INTRODUCTION

A dead variable is that which is defined but not used in the program. Hence it can be removed. A loop-dead variable is that which is defined inside the loop but not used in any operation inside the loop. Since it can be used outside the loop it cannot be removed. However we would like to move its definition outside the loop. Let us see how we can make optimization in such cases.

Program1
```
scanf(all variables);
do
{    t = t * a;
     y = t + b;
     b = a / c;
     c = t - b;
 } while (a > p);
printf(all variables);
```

Optimized Program
```
scanf(all variables);
do
{    t = t * a;
     t1 = t; b1 = b;
     b = a / c;
     c = t - b;
}while (a > p);
y = t1 + b1;
printf(all variables);
```

In the above example we see "y" is defined in the loop but not used in any operation inside the loop. But the value of "y" cannot be neglected as it is used in the program later [in printf(all variables)]. We calculated the value outside the loop by using the auxiliary variables "t1" and "b1", which store the values of "t" and "b" at that position. In the optimized program one arithmetic operation is reduced inside the loop. Though two assignment operations are added but they are cheaper than the single arithmetic operation. Additionally two auxiliary variables are introduced. Auxiliary variables are defined as follows:

Auxiliary variable: Few variables are used in the definition of the loop-dead variables. The values of the loop-dead variables are calculated outside the loop. At that place the value of variables (which are used in the definition of the loop-dead variable) may be different. Hence the value of these variables is copied in some other variables, which are called auxiliary variables.

However the above method is not applicable in the following program.

Program2

```
scanf(all);
while (a >  p)
{
     t  =  t  *  a;
     y  =  t  +  b;
     b  =  a  /  c;
     c  =  t  -  b;
 } while (a >  p);
printf(all);
```

Optimized Program

```
scanf(all);
flag  =  0;
while (a >  p)
{   flag  =  1;
     t  =  t  *  a;
     t1  =  t;  b1  =  b;
     b  =  a  /  c;
     c  =  t  -  b;
} while (a >  p);
If (flag  !=  0)  y  =  t1  +  b1;
printf(all);
```

Above program is same as the first program except in the place of do-while loop we have used while loop. Hence "y=t+b" may not be executed (if the program does not enter while loop). In this case the value of "y" will remain unchanged. Hence we define a flag variable named "flag" to record whether the program enters the loop or not. The initial value of the "flag" is zero and when the program enters the loop it takes the value 1. Flag variables are defined as follows.

Flag Variable: These are used to record at what places in the program the control has reached. These are also record before reaching some place in the program, control reached at what other places.

Program 3

```
scanf(all);
while (a >  p)
{  t  =  a  *  t;
   a  =  t  -  k;
   if (t >  a)
       x  =  t  -  a;
   if (a >  p)
       y  =  b  /  k;
   if (t >  p)
       x =  a  *  d;
   k  =  a  -  t;
}
printf(all);
```

Optimized program

```
scanf(all);
flag1  =  0;  flag2  =  0;
while (a >  p)
{  t  =  a  *  t;
   a  =  t  -  k;
   if (t >  a)
   {  flag1 =  1; t1 =  t; a1 =  a;  }
   if (a >  p)
   {  flag2 =  1; b1 =  b;  k1 =  k;  }
   if (t >  p)
   {  flag1 =  2; a1 =  a; d1 =  d;  }
   k  =  a  -  t;
}
if (flag1 ==    1)   x  =  t1  -  a1;
if (flag1 ==    2)   x  =  a1  *  d1;
if (flag2 ==    1)    y  =  b1  /  k1;
printf(all);
```

In this example "x" and "y" are both defined inside "if" statement of the loop but not used in any operation inside the loop. Here we define two flag variables "flag1" and "flag2" for the definition of each loop-dead variable respectively and also define the auxiliary variables "a1", "b1", "t1", "d1" and "k1" which are used in the definition of the loop-dead variables. Since there are two definitions of "x" and so corresponding flag (flag1) takes two values (one for each definition of x). Every definition of the loop dead variable is given a unique identifier (definition number). Variable "a" is used in both definitions of the loop-dead variable "x", but one time one value of "x" occurs and so we take same auxiliary variable "a1" to store its value.

Program 4

```
scanf(all);
while (a >  p)
{  t  =  t  *  a;
   a  =  t  -  k;
   if (t >  a)
       x  =  t  -  a;
   if (a >  p)
       y  =  b  /  t;
   k  =  a  -  t;
}
printf(all);
```

Optimized program

```
scanf(all);
flag1  =  0;  flag2  =  0;
while (a >  p)
{  t  =  t  *  a;
   a  =  t  -  k;
   if (t >  a)
   {  flag1 =  1; t1 =  t; a1 =  a;  }
   if (a >  p)
   {  flag2 =  1; b1 =  b;  t2 =  t;  }
   k  =  a  -  t;
}
if (flag1 ==    1)   x  =  t1  -  a1;
if (flag2 ==    1)   y  =  b1  /  t2;
printf(all);
```

In the above program the variable "t" is used in the definitions of two different loop-dead variables and so different auxiliary variables "t1" and "t2" are used.

Dependent loop-dead variables

In the previous examples loop dead variables were independent. Now we consider an example where dependent loop-dead variables exist.

Program 5

```
scanf(all);
while (a >  p)
{  a  =  a  *  p;
   if (k >  p)
       x  =  a  /  t;
   k  =  m  +  t;
   if (m >  n)
       y  =  m  +  x;
   m  =  n  *  k;
   t  =  t  +  n;
}
printf(all);
```

Optimized program

```
scanf (all);
flag1 =  0; flag2 =  0; flag21 =  0;
while (a >  p)
{  a  =  a  *  p;
   if (k >  p)
   {  flag1 =  1; a1 =  a; t1 =  t;  }
   k  =  m  +  t;
   if (m >  n)
  {  flag2=    1;  flag21=    flag1;  m1=    m;  t21=    t1;
     a21=  a1;}
   m  =  n  *  k;
   t  =  t  +  n;
}
if (flag21  ==    1)    x  =  a21  /  t21;
if (flag2  ==    1)   y  =  m1  +  x;
if (flag  ==  1)   x  =  a1  /  t1;
printf(all);
```

In this example, "y" is a loop-dead variable as it is defined in the loop but not used in any operation inside the loop. Although "x" is used in the loop, yet only in the definition of the loop-dead variable "y" and so it is also a loop-dead variable and the loop-dead variable "y" is dependent on the loop-dead variable "x" and so it is dependent loop-dead variable. To store the value of "x" at the position (when "y" is defined) we define two auxiliary variables "a21" and "t21" (in the definition of "y"). They are used (outside the loop) to calculate the value of "x" at the definition of "y". We also need to define another flag variable "flag21" which signifies, what was the value of "flag1" when "y" was defined.

This example leads to concept of introducing flag and auxiliary variables in an appropriate manner, which we shall discuss in the next section. From this example we conclude more powerful definition of the loop-dead variable as:

Definition: A variable "z" is called loop-dead variable if "z" is not used in any operation inside the loop except in the definition of other loop-dead variables.

2. NOMENCLATURE

In the previous section we discussed the definition of the loop-dead variable and some optimized programs. When the loop-dead variables are independent then there is no problem in writing the names of the flag and auxiliary variables. However when the loop-dead variables are dependent then names of these variables are determined by inter dependence of various loop-dead variables. In this section we discuss a method to write the names of the flag and auxiliary variables.

Names of the flag and auxiliary variables

Following method is used to find names of the flag and auxiliary variables in the loop.

Let "α" be a non loop-dead variable and "β", "γ" and "δ" be the loop-dead variables. Let "α" is used in some definition of "β", which is used in some definition of "γ", which is used in some definition of "δ". Hence "α_β", "$\alpha_{\beta\gamma}$" and "$\alpha_{\beta\gamma\delta}$" are the auxiliary and "f_β", "$f_{\beta\gamma}$" and "$f_{\beta\gamma\delta}$" are the flag variables. They are define as follows:

i) $\alpha_\beta = \alpha$ and $f_\beta =$ definition number of "β", are written in the definition of "β".
ii) $\alpha_{\beta\gamma} = \alpha_\beta$ and $f_{\beta\gamma} = f_\beta$ are written in the definition of "γ".
iii) $\alpha_{\beta\gamma\delta} = \alpha_{\beta\gamma}$ and $f_{\beta\gamma\delta} = f_{\beta\gamma}$ are written in the definition of "δ".
iv) The generalization of above is done.

Program 6

```
scanf(all);
while (x >  a)
{ t  =  a  *  n;
   if (k >  p)
       x  =  a  /  t;
   a  =  t  -  k;
   if (p >  c)
       y  =  p  *  x;
   k  =  k  *  c;
   q  =  a  +  n;
   if (q >  b)
        z  =  y  *  x;
   p  =   q  *  c;
}

printf(all);
```

Optimized program

```
scanf(all);
$f_x$ = 0; $f_y$ = 0; $f_z$ = 0;
$f_{xy}$ = 0; $f_{yz}$ = 0; $f_{xz}$ = 0;  $f_{xyz}$ = 0;
while (x >  a)
{  t  =  a  *  n;
    if (k >  p)
    {  $f_x$ = 1; $a_x$ = a; $t_x$ = t;  }
    a  =  t  -  k;
    if (p >  c)
   {   $f_y$ = 1; $f_{xy}$ = $f_x$; $p_y$ = p; $a_{xy}$ = $a_x$; $t_{xy}$ = $t_x$; }
    k  =  k  *  c;
    q  =  a  +  n;
    if (q >  b)
   {  $f_z$ = 1; $f_{yz}$ = $f_y$; $f_{xz}$ = $f_x$; $f_{xyz}$ = $f_{xy}$;
      $a_{xyz}$ = $a_{xy}$; $t_{xyz}$ = $t_{xy}$; $p_{yz}$ = $p_y$; $a_{xz}$ = $a_x$; $t_{xz}$ = $t_x$;  }
   p  =  q * c;
}
if ($f_{xyz}$ ==  1)      xyz  =  $a_{xyz}$  /  $t_{xyz}$;
if ($f_{xyz}$ ==  0)      xyz  =  x;
if ($f_{xz}$ ==  1)        xz  =  $a_{xz}$  /  $t_{xz}$;
if ($f_{xz}$ ==  0)        xz  =  x;
if ($f_{yz}$ ==  1)        yz  =  $p_{yz}$  *  xyz;
if ($f_{yz}$ ==  0)        yz  =  y;
if ($f_z$ ==  1)          z  =  yz  *  xz;
if ($f_{xy}$ ==  1)        xy  =  $a_{xy}$  /  $t_{xy}$;
if ($f_{xy}$ ==  0)        xy  =  x;
if ($f_y$ ==  1)          y  =  $p_y$ *  xy;
if ($f_x$ ==  1)          x  =  $a_x$  /  $t_x$;
printf(all);
```

3. DEADNESS GRAPH

It shows the relationship between the loop-dead variables and the variables, which are used in its definition. This graph is a cycle free graph and every vertex of the graph is either a loop-dead variable or a variable used in the definition of the loop-dead variable. This graph shows dependence of one loop-dead variable on another.

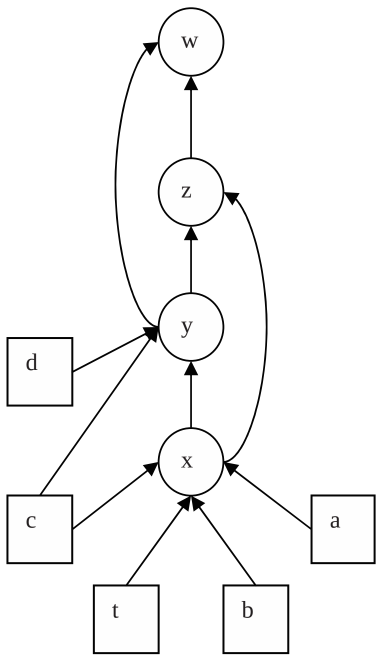
Algorithm to draw the graph

- i) Make a circle (as vertex) corresponding to every loop-dead variable and a square (as vertex) corresponding to every variable used in the definition of the loop-dead variables.
- ii) Draw a directed edge from "x" to "y", if "x" is used in the definition of the loop-dead variable "y". "x" may be loop dead variable or the variable used in the definition of loop-dead variables.
- iii) For convenience we place the independent loop-dead variable at the bottom of the graph and arrange the other loop-dead variables from lower to upper depending on their dependencies.

Now we take an example and draw the corresponding deadness graph.

Program 7

```
scanf(all);
while (a >  p)
{  if (k >  p)  x = a  /  t;
   if (a >  c)   x =  b  -  c;
   a  =  t  -  k;
   if (p >  k)   y =  d  *  x;
   if (k >  a)   y =  c  *  x;
   k  =  c  +  a;
   if (b >  a)   z =  y  /  x;
   b  =  t  -  k;
   if (p >  b)   w =  z  +  y;
   c  =  c  *  t;
   p  =  k  *  c;
}
printf(all);
```



Names of the flag variables and auxiliary variables

- i) For every path "αp", where "α" is square vertex (i.e. "α" is not a loop dead variable), we define the corresponding auxiliary variable "α_p".
- ii) For every path "p" starting from the loop dead variable we define flag variable as "f_p". The flag variable "f_p" is independent if the length of path "p" is 1. Otherwise the flag variable is dependent.

From the deadness graph it can be concluded that the auxiliary variables used in the program are a_x, t_x, b_x , c_x, d_y, c_y, a_{xy}, b_{xy}, c_{xy}, t_{xy}, a_{xyz}, b_{xyz}, c_{xyz}, t_{xyz}, a_{xz}, b_{xz}, c_{xz}, t_{xz}, d_{yz}, c_{yz}, a_{xyzw}, b_{xyzw}, c_{xyzw}, t_{xyzw}, a_{xzw}, b_{xzw}, c_{xzw}, t_{xzw}, a_{xyw}, b_{xyw}, c_{xyw}, t_{xyw}, d_{yzw}, c_{yzw}, d_{yw}, c_{yw} and the flag variables are f_x, f_y, f_{xy}, f_z, f_{xz}, f_{xyz}, f_{yz}, f_w, f_{xyzw}, f_{xyw}, f_{xzw}, f_{yzw}, f_{yw}, $f_{zw.}$ Among these independent flag variables are $f_{x,}$,f_y, f_z and f_w and others are dependent flag variables.

4. ALGORITHM

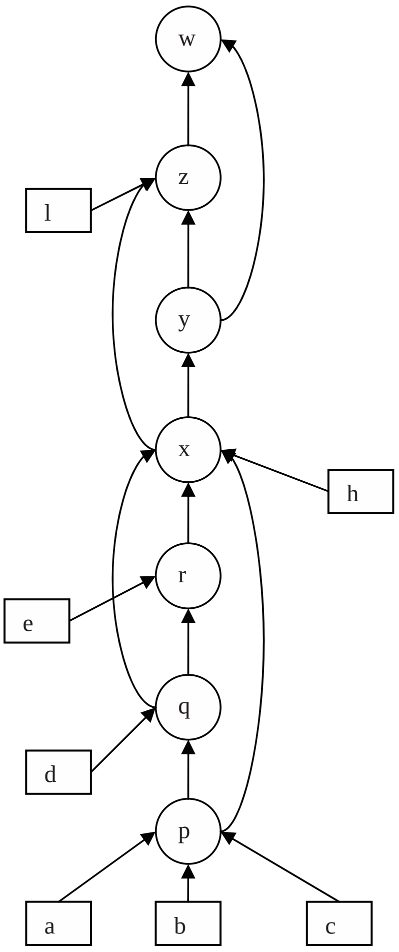
In the previous sections we saw how to name the flag and auxiliary variables from the deadness graph of a loop. In this section we will see how a definition of a loop-dead variable will be moved out of the loop.

Algorithm

1. Make the deadness graph corresponding to the loop.
2. Initialize every flag variables as "zero".
3. Let A= B op C be the r^{-th} definition of loop dead variable A, it is replace by the following assignment statements inside the loop.
    a. f_A= r.
    b. $K_{\alpha A} = K_\alpha$ for every path "Kα" start from a non loop-dead variable "K" and end at "B" (or "C").
    c. $f_{\beta A} = f_\beta$ for every path "β" start from some loop-dead variable and end at "B" (or "C").
4. For above definition (A= B op C) following computations are put after the loop.
   If ($f_{A\gamma}$ == r) Aγ = Bγ op Cγ, for every path "Aγ".
   [We have used subscript as notation for "B" (or "C") if "B"(or "C") is not a loop-dead variable and it will be written as "B_γ"(or "C_γ")].
5. If ($f_{A\gamma}$ == 0) Aγ = A for every path Aγ.
6. Using steps 3 and 4 we find out assignment statements inside the loop and corresponding loop-dead computations at the end of the loop for all definitions of the loop-dead variables.
7. In the loop-dead computation at the end of the loop, we compute the operations of independent flag variables at the end and among the dependent flag variables we compute the operation of those flag variables whose paths start from lower vertex (circular vertex) and then compute others correspondingly lower to upper vertices (circular vertex) of the deadness graph.

In step 3 of above algorithm the loop-dead computation is replaced by assignment statements of the flag and auxiliary variables. The step 4 and 5 move the loop dead computation outside the loop. Step 7 insures the sequencing of the loop-dead computation at the end of the loop.

To make the program more optimized (in step 2) we give the initial value of independent flag variables as "zero" and dependent flag variables as "–1". It can be understood by following example. Let in program 6, y = p * x is never executed but z = y * x is executed. Now f_{xyz} will be zero but statement xyz = x is useless. Similarly f_{xy} will be zero but statement xy = x is useless.



Deadness graph

Let deadness graph of a loop is given as above and x = h * q + r be the kth definition of the loop-dead variable "x". From this let us find the assignment statements of the flag and auxiliary variables inside the loop and loop-dead computations at the end of the loop.

To get assignment statement of the flag variables, we find paths, which start from some loop-dead variable and end at "q" or "r". These are r, q-r, q, p-q and p-q-r. Corresponding flag variables are "f_{rx}", "f_{qrx}", "f_{qx}", "f_{pqx}" and "f_{pqrx}" and assignment statements are: f_{rx}= f_r, f_{qrx}= f_{qr}, f_{qx}= f_q, f_{pqx}= f_{pq} and f_{pqrx}= f_{pqr}. One more assignment is f_x= k (from 3(a)).

To get assignment statements of the auxiliary variables, we find paths, which start from a non loop-dead variable and end at "h", "q" or "r". These are h, a-p-q, b-p-q, c-p-q, d-q, a-p-q-r, b-p-q-r, c-p-q-r, d-q-r and e-r. Corresponding assignment statements are h_x= h, a_{pqx}= a_{pq}, b_{pqx}= b_{pq}, c_{pqx}= c_{pq}, d_{qx}= d_q, a_{pqrx}= a_{pqr}, b_{pqrx}= b_{pqr}, c_{pqrx}= c_{pqr}, d_{qrx}= d_{qr}, e_{rx}= e_r. To find loop-dead computations outside the loop, we find paths, which start from the loop-dead variable "x". These paths are x, x-y, x-y-z, x-z, x-y-w, x-z-w and x-y-z-w and corresponding computations are

```
if($f_x$==k)      x= $h_x$*qx+ rx;
if($f_{xy}$==k)     xy= $h_{xy}$*qxy+ rxy;
if($f_{xyz}$==k)    xyz= $h_{xyz}$*qxyz+ rxyz;
if($f_{xz}$==k)     xz= $h_{xz}$*qxz+ rxz;
if($f_{xyw}$==k)    xyw= $h_{xyw}$*qxyw+ rxyw;
if($f_{xzw}$==k)    xzw= $h_{xzw}$*qxzw+ rxzw;
if($f_{xyzw}$==k)   xyzw= $h_{xyzw}$*qxyzw+ rxyzw;
```

But in the complete optimized program we will calculate the value of the loop-dead variable "x" at the end (as stated in the step 7 of the algorithm).

Using the above algorithm the optimized version of program 7 is obtained as follows:

```
scanf(all);
$f_x$= 0; $f_y$= 0; $f_z$= 0; $f_w$= 0;
$f_{xy}$= –1; $f_{yz}$= –1; $f_{xz}$= –1; $f_{yw}$= –1; $f_{zw}$= –1;    $f_{xyz}$= –1; $f_{xzw}$= –1; $f_{xyw}$= –1; $f_{yzw}$= –1;  $f_{xyzw}$= –1;
while(a> p)
{  a= a*t;
   if(k> p)  { $a_x$= x; $t_x$= t; $f_x$= 1; }
   if(a> c)  {  $b_x$= x; $c_x$= c; $f_x$= 2; }
   a=t-k;
   if(p> q)  { $d_y$= d; $a_{xy}$= $a_x$; $b_{xy}$= $b_x$; $c_{xy}$= $c_x$; $t_{xy}$= $t_x$; $f_{xy}$= $f_x$; $f_y$= 1; }
   if(k> a)  { $c_y$= c; $a_{xy}$= $a_x$; $b_{xy}$= $b_x$; $c_{xy}$= $c_x$; $t_{xy}$= $t_x$; $f_{xy}$= $f_x$; $f_y$= 2; }
   k= c+ a;
   if(b> a)  {  $a_{xyz}$= $a_{xy}$; $b_{xyz}$= $b_{xy}$; $c_{xyz}$= $c_{xy}$; $t_{xyz}$= $t_{xy}$;
                $a_{xz}$= $a_x$; $b_{xz}$= $b_x$; $c_{xz}$= $c_x$; $t_{xz}$= $t_x$;  $d_{yz}$= $d_y$; $c_{yz}$= $c_y$;
                $f_{xyz}$= $f_{xy}$; $f_{xz}$= $f_x$; $f_{yz}$= $f_y$; $f_z$= 1;                    }
   if(p> b)  { $a_{xyzw}$= $a_{xyz}$; $b_{xyzw}$= $b_{xyz}$; $c_{xyzw}$= $c_{xyz}$; $t_{xyzw}$= $t_{xyz}$;
                $a_{xzw}$= $a_{xz}$; $b_{xzw}$= $b_{xz}$; $c_{xzw}$= $c_{xz}$; $t_{xzw}$= $t_{xz}$;
                $a_{xyw}$= $a_{xy}$; $b_{xyw}$= $b_{xy}$; $c_{xyw}$= $c_{xy}$; $t_{xyw}$= $t_{xy}$;
                $d_{yzw}$= $d_{yz}$; $c_{yzw}$= $c_{yz}$;  $d_{yw}$= $d_y$; $c_{yw}$= $c_y$;
                $f_{xyzw}$= $f_{xyz}$; $f_{xyw}$= $f_{xy}$; $f_{xzw}$= $f_{xz}$; $f_{yzw}$= $f_{yz}$;
                $f_{yw}$= $f_y$; $f_{zw}$= $f_z$; $f_w$ = 1;                          }

   c= c*t;
   p= k*c;
}
if($f_{xyzw}$==2)   xyzw=$b_{xyzw}$-$c_{xyzw}$
if($f_{xzw}$==2)    xzw=$b_{xzw}$-$c_{xzw}$;
if($f_{xyw}$==2)    xyw=$b_{xyw}$-$c_{xyw}$;
if($f_{xyz}$==2)    xyz=$b_{xyz}$-$c_{xyz}$;
if($f_{xz}$==2)    xz=$b_{xz}$-$c_{xz}$;
if($f_{xy}$==2)    xy=$b_{xy}$-$c_{xy}$;
if($f_{xyzw}$==1)   xyzw=$a_{xyzw}$/$t_{xyzw}$;
if($f_{xzw}$==1)    xzw=$a_{xzw}$/$t_{xzw}$;
if($f_{xyw}$==1)    xyw=$a_{xyw}$/$t_{xyw}$;
if($f_{xyz}$==1)    xyz=$a_{xyz}$/$t_{xyz}$;
if($f_{xz}$==1)     xz=$a_{xz}$/$t_{xz}$;
if($f_{xy}$==1)     xy=$a_{xy}$/$t_{xy}$;
if($f_{xyzw}$==0)    xyzw=x;
if($f_{xzw}$==0)     xzw=x;
if($f_{xyw}$==0)     xyw=x;
if($f_{xyz}$==0)     xyz=x;
if($f_{xz}$==0)      xz=x;
if($f_{xy}$==0)      xy=x;
if($f_{yzw}$==2)     yzw=$c_{yzw}$*xyzw;
if($f_{yw}$==2)      yw=$c_{yw}$*xyw;
if($f_{yz}$==2)      yz=$c_{yz}$*xyz;
if($f_{yzw}$==1)     yzw=$d_{yzw}$*xyzw;
if($f_{yw}$==1)      yw=$d_{yw}$*xyw;
if($f_{yz}$==1)      yz=$d_{yz}$*xyz;
if($f_{yzw}$==0)     yzw=y;
if($f_{yw}$==0)      yw=y;
if($f_{yz}$==0)      yz=y;
if($f_{zw}$==1)      zw=$y_{zw}$/$x_{zw}$;
if($f_{zw}$==0)      zw=z;
if($f_x$==2)       x=$b_x$-$c_x$;
if($f_x$==1)       x=$a_x$/$t_x$;
if($f_y$==2)       y=$c_y$*xy;
if($f_y$==1)       y=$d_y$*xy;
if($f_z$==1)       z=yz/xz;
if($f_w$==1)       w=zw/yw;
```

```
x= a/t;
x= b-c;
```

```
y= d*x;
y= c*x;
```

```
z= y/x;
```

```
w= z+ y
```

REFERENCES

[1] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman, Compiler: Principles, Techniques, and Tools, 1986, Addison-Wesley.

[2] Steve Muchnik, Advance Compiler Design and Implementation, 1997, Morgan Kaufmann.