Demo of Aligator

Loading the package

```
In[1]:= << Aligator.m
```

```
Aligator.m
Automated Loop Invariant Generation by Algebraic Techniques Over the Rationals.
Package written by Laura Kovacs – © RISC Linz and EPFL Lausanne – V 0.3 (2008-02-01)
```

```
Dependencies.m by Manuel Kauers and Burkhard Zimmermann. Changed by Laura Kovacs for
Invariant Generation – © RISC Linz – V 0.27 (2008-02-07)
```

```
Fast Zeilberger Package by Peter Paule, Markus Schorn, and Axel Riese – © RISC Linz – V
3.52 (01/12/05)
```

Examples for Loops without Conditionals

Note: * stand for own examples

Simple Example 0 *

```
(*
 WHILE[…, x:=x+1]
*)
```

```
In[2]:= Aligator[WHILE[…, x := x + 1]] // Timing

        Method is complete!

Out[2]= {0.078 Second, True}
```

Simple Example 1 *

```
(*
 x:=0;y:=1;
 WHILE[…, x:=2*x;y:=1/2*y+1]
*)
```

In[3]:=
```
Aligator[WHILE[…, x := 2 * x; y := 1 / 2 * y + 1]] // Timing
```

```
Method is complete!
```

Out[3]=
```
{0.172 Second, x (-2 + y) == x[0] (-2 + y[0])}
```

In[47]:=
```
Aligator[WHILE[…, x := 2 * x; y := 1 / 2 * y + 1], IniVal → {x := 0; y := 1}] // Timing
```

```
Method is complete!
```

Out[47]=
```
{0.125 Second, x y == 2 x}
```

Simple Example 2 *

```
(*
 a:=0;b:=0;
 WHILE[…, a:=a+2*b+1; b:=b+1]
*)
```

In[4]:=
```
Aligator[WHILE[…, a := a + 2 * b + 1; b := b + 1]] // Timing
```

```
Method is complete!
```

Out[4]=
```
{0.032 Second, a + b[0]^2 == b^2 + a[0]}
```

```
Aligator[WHILE[…, a := a + 2 * b + 1; b := b + 1], IniVal → {a := 0; b := 0}]
```

```
Method is complete!
```

$$a == b^2$$

Simple Example 3 *

```
(*
 a:=0;b:=0;c:=1;
 WHILE[…,a:=a+1;b:=b+c;c:=c+2]
*)
```

In[5]:=
```
Aligator[WHILE[…, a := a + 1; b := b + c; c := c + 2]] // Timing
```

```
Method is complete!
```

Out[5]=
```
{0.062 Second, 2 a + c[0] == c + 2 a[0] && 4 b + 2 c + c[0]^2 == c^2 + 4 b[0] + 2 c[0]}
```

In[6]:=
```
Aligator[WHILE[…, a := a + 1; b := b + c; c := c + 2], IniVal → {a := 0; b := 0; c := 1}] // Timi
```

```
Method is complete!
```

Out[6]=
```
{0.063 Second, 1 + 2 a == c && 4 b == (-1 + c)^2}
```

Simple Example 4 *

```
(*
 a:=0;b:=0;c:=1;s:=0;
 WHILE[…, a:=a+1; b:=b+c; c:=c+2; s:=s+2*a+1]
*)
```

```
In[7]:= Aligator[WHILE[…, a := a + 1; b := b + c; c := c + 2; s := s + 2 * a + 1]] // Timing
```

```
Method is complete!
```

```
Out[7]= {0.156 Second, 2 a + c[0] == c + 2 a[0] &&
  2 b + c (3 + 2 a[0]) + c[0]^2 + 2 s[0] == 2 s + 2 b[0] + (3 + c + 2 a[0]) c[0] &&
  c^2 + 4 c (1 + a[0]) + c[0]^2 + 4 s[0] == 4 s + 2 (2 + c + 2 a[0]) c[0]}
```

```
In[48]:= Aligator[WHILE[…, a := a + 1; b := b + c; c := c + 2; s := s + 2 * a + 1],
           IniVal → {a := 0; b := 0; c := 1; s := 0}] // Timing
```

```
Method is complete!
```

```
Out[48]= {0.094 Second, 1 + 2 a == c && b + c == 1 + s && c (2 + c) == 3 + 4 s}
```

Division Example, by E. W. Dijkstra

```
(*
 rem:=x;quo:=0;
 WHILE[…,rem:=rem-y;quo:=quo+1]
*)
```

```
In[8]:= Aligator[WHILE[…, rem := rem - y; quo := quo + 1]] // Timing
```

```
Method is complete!
```

```
Out[8]= {0.047 Second, rem + quo y == y quo[0] + rem[0]}
```

```
In[9]:= Aligator[WHILE[…, rem := rem - y; quo := quo + 1], IniVal → {rem := x; quo := 0}] // Timing
```

```
Method is complete!
```

```
Out[9]= {0.031 Second, rem + quo y == x}
```

Integer Square Root, by M. Kirchner - version 1

```
(*
 k:=0; j:=1; m:=1;
 WHILE[m≤n, k:=k+1; j:=j+2; m:=m+j]
*)
```

```
In[10]:= Aligator[WHILE[m ≤ n, k := k + 1; j := j + 2; m := m + j]] // Timing
```

```
Method is complete!
```

```
Out[10]= {0.078 Second, j + 2 k[0] == 2 k + j[0] && 2 j + j^2 + 4 m[0] == 4 m + j[0] (2 + j[0])}
```

```
In[11]:= Aligator[WHILE[m ≤ n, k := k + 1; j := j + 2; m := m + j], IniVal → {k := 0; j := 1; m := 1}]
```

```
Method is complete!
```

```
Out[11]= {0.063 Second, j == 1 + 2 k && (1 + j)^2 == 4 m}
```

Integer Square Root, by D. E. Knuth - version 2

```
(*
 y:=a/2;r:=0;
 WHILE[…,y:=y-r; r:=r+1]
*)
```

■ Top level command: Aligator

```
In[12]:= Aligator[WHILE[…, y := y - r; r := r + 1]] // Timing
```

```
Method is complete!
```

```
Out[12]= {0.047 Second, r^2 + 2 y + r[0] == r + r[0]^2 + 2 y[0]}
```

```
In[13]:= Aligator[WHILE[…, y := y - r; r := r + 1], IniVal → {y := a / 2; r := 0}] // Timing
```

```
Method is complete!
```

```
Out[13]= {0.046 Second, a + r == r^2 + 2 y}
```

Integer Cubic Root, by D. E. Knuth

```
(*
 x:=a;r:=1;s:=13/4;
 WHILE[…,x:=x-s; s:=s+6*r+3; r:=r+1]
*)
```

```
In[14]:= Aligator[ WHILE[…, x := x - s; s := s + 6 * r + 3; r := r + 1]] // Timing
```

```
Method is complete!
```

```
Out[14]= {0.125 Second, 3 r^2 + s[0] == s + 3 r[0]^2 &&
  r + 2 r^3 + 2 x + r[0]^2 (3 + 4 r[0]) + 2 r s[0] == 3 r^2 + r[0] + 6 r r[0]^2 + 2 r[0] s[0] + 2 x[0]}
```

```
In[15]:= Aligator[ WHILE[…, x := x - s; s := s + 6 * r + 3; r := r + 1],
    IniVal → {x := a; r := 1; s := 13 / 4}] // Timing
```

```
Method is complete!
```

```
Out[15]= {0.094 Second, 1/4 + 3 r^2 == s && 1 + 4 a + 6 r^2 == 3 r + 4 r^3 + 4 x}
```

Consecutive Cubes, by H. Cohen

```
(*
 n:=0; x:=0; y:=1; z:=6;
WHILE[(n≤N), x:=x+y; y:=y+z; z:=z+6; n:=n+1 ]
*)
```

```
In[16]:= Aligator[ WHILE[(n ≤ N), x := x + y; y := y + z; z := z + 6; n := n + 1 ]] // Timing
```

```
Method is complete!
```

```
Out[16]= {0.172 Second, 6 n + z[0] == z + 6 n[0] &&
    2 n^3 + 2 x[0] + 2 n (2 + 6 n[0] + 3 n[0]^2 + y[0]) + n^2 z[0] + n[0] (1 + n[0]) z[0] ==
     6 n^2 (1 + n[0]) + 2 (x + n[0] (2 + 3 n[0] + n[0]^2 + y[0])) + n (1 + 2 n[0]) z[0] &&
    3 n^2 + 3 n[0] + 3 n[0]^2 + y[0] + n (-3 - 6 n[0] + z[0]) == y + n[0] z[0]}
```

```
In[17]:= Aligator[ WHILE[(n ≤ N), x := x + y; y := y + z; z := z + 6; n := n + 1 ],
           IniVal → {n := 0; x := 0; y := 1; z := 6}] // Timing
```

```
Method is complete!
```

```
Out[17]= {0.094 Second, n^3 == x && 1 + 3 n + 3 n^2 == y && 6 + 6 n == z}
```

Sum of powers ^5, by M. Petter

```
(*
 x:=0; y:=0;
WHILE[(y≠k), x:=x+y^5; y:=y+1]
*)
```

```
In[18]:= Aligator[ WHILE[(y ≠ k), x := x + y^5; y := y + 1]] // Timing
```

```
Method is complete!
```

```
Out[18]= {0.156 Second, 12 x + y^2 + 6 y^5 + 5 y[0]^4 + 2 y[0]^6 == 5 y^4 + 2 y^6 + 12 x[0] + y[0]^2 + 6 y[0]^5}
```

```
In[19]:= Aligator[ WHILE[(y ≠ k), x := x + y^5; y := y + 1], IniVal → {x := 0; y := 0}] // Timing
```

```
Method is complete!
```

```
Out[19]= {0.125 Second, 12 x + y^2 + 6 y^5 == 5 y^4 + 2 y^6}
```

Fibonacci Numbers * (based on the book of D. E. Knuth)

```
(*
 r:=1;q:=1;
 WHILE[…, t:=r; r:=r+q; q:=t]
*)
```

```
In[20]:= Aligator[WHILE[…, t := r; r := r + q; q := t]] // Timing
```

```
Method is complete!
```

```
Out[20]= {0.39 Second, q^4 + 2 q^3 r + r^4 == q^2 r^2 + 2 q r^3 + (q[0]^2 + q[0] r[0] - r[0]^2)^2}
```

```
In[21]:= Aligator[ WHILE[…, t := r; r := r + q; q := t], IniVal → {r := 1; q := 1}] // Timing
```

```
Method is complete!
```

```
Out[21]= {0.219 Second, q^4 + 2 q^3 r + r^4 == 1 + q^2 r^2 + 2 q r^3}
```

Fibonacci - Type Example * (based on the book of R. Stansley)

```
(*
 r:=2;q:=1;x:=2;
 WHILE[…, t:=r; r:=2 r- 8*q; q:=t;x:=8*x]
*)
```

```
In[22]:= Aligator[ WHILE[…, t := r; r := 2 r - 8 * q; q := t; x := 8 x]] // Timing
```

```
Method is complete!
```

```
Out[22]= {0.281 Second, x (8 q[0]^2 - 2 q[0] r[0] + r[0]^2) == (8 q^2 - 2 q r + r^2) x[0]}
```

```
In[23]:= Aligator[ WHILE[…, t := r; r := 2 r - 8 * q; q := t; x := 8 x],
           IniVal → {r := 2; q := 1; x := 2}] // Timing
```

```
Method is complete!
```

```
Out[23]= {0.203 Second, 8 q^2 + r^2 == 2 q r + 4 x}
```

Examples for Loops with Conditionals

Note: * stand for own examples

Simple Example 1 *

```
(*
 WHILE[…,
      IF[…, a:=a/2;b:=b/2;c:=c+1;d:=d+1,
            a:=a/4;b:=b/4]]
*)
```

```
In[24]:= Aligator[WHILE[…,
             IF[…, a := a / 2; b := b / 2; c := c + 1; d := d + 1,
                   a := a / 4; b := b / 4]]] // Timing
```

```
Method is complete!
```

```
Out[24]= {0.219 Second, a b[0] == b a[0] && c + d[0] == d + c[0]}
```

```
In[25]:= Aligator[WHILE[…, IF[…, a := a / 2; b := b / 2; c := c + 1; d := d + 1, a := a / 4; b := b / 4]],
           IniVal → {a := 1; b := 2; c := 0; d := 0}] // Timing
```

```
Method is complete!
```

```
Out[25]= {0.219 Second, 2 a == b && c == d}
```

Simple Example 2 *

```
(*
 a:=1;b:=1;c:=2;
 While[…,
  IF[…, a:=a-1; b:=b+2; c:=c-1,
        a:=a+1; b:=b-1; c:=c-2]]
*)
```

```
In[26]:= Aligator[WHILE[…,
           IF[…, a := a - 1; b := b + 2; c := c - 1,
                 a := a + 1; b := b - 1; c := c - 2]]] // Timing
```

```
Method is complete!
```

```
Out[26]= {0.109 Second, 5 a + 3 b + c == 5 a[0] + 3 b[0] + c[0]}
```

```
In[27]:= Aligator[WHILE[…,
           IF[…, a := a - 1; b := b + 2; c := c - 1,
                 a := a + 1; b := b - 1; c := c - 2]], IniVal → {a := 1; b := 1; c := 2}] // Timing
```

```
Method is complete!
```

```
Out[27]= {0.11 Second, 5 a + 3 b + c == 10}
```

Simple Example 3 *

```
(*
 WHILE[…,
   IF[…, x:=x+1; y:=y+1,
         x:=x-z; y:=y+z^2;z:=z-1]]
*)
```

```
In[28]:= Aligator[WHILE[…,
            IF[…, x := x + 1; y := y + 1,
                  x := x - z; y := y + z^2; z := z - 1]]] // Timing
```

```
Method is complete!
```

```
Out[28]= {0.156 Second, 3 y + 2 z + 3 z^2 + z^3 + 3 x[0] == 3 x + 3 y[0] + z[0] (2 + 3 z[0] + z[0]^2)}
```

```
In[29]:= Aligator[WHILE[…,
            IF[…, x := x + 1; y := y + 1,
                  x := x - z; y := y + z^2; z := z - 1]], IniVal → {x := 1; y := 2; z := 0}] // Timing
```

```
Method is complete!
```

```
Out[29]= {0.156 Second, 3 (1 + x) == 3 y + z (2 + 3 z + z^2)}
```

Simple Example 4 *

```
(*
 WHILE[x ≥ 0,
     IF[…, x:=x+z; y:=y+1; z:=z-2,
            x:=x+y; y:=y-2]]
*)
```

```
In[30]:= Aligator[WHILE[x ≥ 0,
     IF[…, x := x + z; y := y + 1; z := z - 2,
             x := x + y; y := y - 2]]] // Timing
```

```
Method is complete!
```

```
Out[30]= {0.5 Second, True}
```

```
In[31]:= Aligator[WHILE[x ≥ 0,
     IF[…, x := x + z; y := y + 1; z := z - 2,
             x := x + y; y := y - 2]], IniVal → {x := 1; y := 2; z := 0}] // Timing
```

```
Method is complete!
```

```
Out[31]= {0.485 Second, True}
```

Wensley's Real Division Algorithm, by. B. Wegbreit

```
(*
 a:=0;b:=Q/2;d:=1;y:=0;
 WHILE[d≥Tol,
      IF[P<a+b,
           b:=b/2;d:=d/2,
           a:=a+b;y:=y+d/2;b:=b/2;d:=d/2
        ]
 ]
*)
```

```
In[49]:= Aligator[WHILE[d ≥ Tol,
          IF[P < a + b,
               b := b / 2; d := d / 2,
               a := a + b; y := y + d / 2; b := b / 2; d := d / 2
            ]]] // Timing
```

```
Method is complete!
```

```
Out[49]= {0.25 Second,
  b d[0] == d b[0] && a d + 2 b y[0] == 2 b y + d a[0] && a d[0] + 2 b[0] y[0] == 2 y b[0] + a[0] d[0]}
```

```
In[50]:= Aligator[WHILE[d ≥ Tol,
              IF[P < a + b,
                   b := b / 2; d := d / 2,
                   a := a + b; y := y + d / 2; b := b / 2; d := d / 2
                ]], IniVal → {a := 0; b := Q / 2; d := 1; y := 0}] // Timing
```

Method is complete!

```
Out[50]= {0.25 Second, 2 b == d Q && a d == 2 b y && a == Q y}
```

Square Root, by K. Zuse

```
(*
   r:=a-1; q:=1; p:=1/2;
   WHILE[( 2*p*r≥err),
       IF[(2*r-2*q*p≥0),
          r:=2*r-2*q-p;q:=q+p;p:=p/2,
          r:=2*r;p:=p/2]]
*)
```

```
In[33]:= Aligator[ WHILE[( 2 * p * r ≥ err),
             IF[(2 * r - 2 * q * p ≥ 0),
                   r := 2 * r - 2 * q - p; q := q + p; p := p / 2,
                   r := 2 * r; p := p / 2]]] // Timing
```

Method is complete!

```
Out[33]= {0.375 Second, q^2 + 2 p r == q[0]^2 + 2 p[0] r[0]}
```

```
In[34]:= Aligator[ WHILE[( 2 * p * r ≥ err),
             IF[(2 * r - 2 * q * p ≥ 0),
                   r := 2 * r - 2 * q - p; q := q + p; p := p / 2,
                   r := 2 * r; p := p / 2]], IniVal → {r := a - 1; q := 1; p := 1 / 2}] // Timing
```

Method is complete!

```
Out[34]= {0.406 Second, a == q^2 + 2 p r}
```

LCM & GCD computation, by E. W. Dijkstra

```
(*
 x:=a;y:=b;u:=b;v:=a;
     WHILE[x≠y,
              IF[x>y,
                       x:=x-y;v:=v+u,
                       y:=y-x; u:=u+v
                ]
      ];
*)
```

```
In[51]:= Aligator[ WHILE[x ≠ y,
                         IF[x > y,
                                  x := x – y; v := v + u,
                                  y := y – x;  u := u + v
                          ]
                ]] // Timing
```

```
        Method is complete!
```

```
Out[51]= {0.438 Second, u x + v y == u[0] x[0] + v[0] y[0]}
```

```
In[52]:= Aligator[ WHILE[x ≠ y,
                         IF[x > y,
                                  x := x – y; v := v + u,
                                  y := y – x;  u := u + v
                          ]
                ], IniVal → {x := a; y := b; u := b; v := a}] // Timing
```

```
        Method is complete!
```

```
Out[52]= {0.437 Second, 2 a b == u x + v y}
```

Binary Product of Two Natural Numbers, by D. E. Knuth

```
        (*
         x:=a; y:=b; z:=0;
           WHILE[( y≠0),
               IF[(Mod[y,2]=1),
                       z:=z+x; y:=y-1];
               x:=2*x; y:=y/2]
        *)
```

```
In[53]:= Aligator[ WHILE[( y ≠ 0),
               IF[(Mod[y, 2] = 1),
                          z := z + x; y := y – 1];
               x := 2 * x; y := y / 2]] // Timing
```

```
        Method is complete!
```

```
Out[53]= {0.203 Second, x y + z == x[0] y[0] + z[0]}
```

```
In[54]:= Aligator[WHILE[( y ≠ 0),
               IF[(Mod[y, 2] = 1),
                          z := z + x; y := y – 1];
               x := 2 * x; y := y / 2], IniVal → {x := a; y := b; z := 0}] // Timing
```

```
        Method is complete!
```

```
Out[54]= {0.219 Second, a b == x y + z}
```

Computing the floor of the square root, by E. W. Dijkstra

```
(* 2nd WHILE*)
(*
 WHILE[(q≠1),
        IF[(r≥h),
           q:=q/4; h:=p+q; p:=p/2; p:=p+q; r:=r-h,
           q:=q/4; h:=p+q; p:=p/2]]
*)
```

```
In[37]:= Aligator[ WHILE[(q ≠ 1),
                  IF[(r ≥ h),
                      q := q / 4; h := p + q; p := p / 2; p := p + q; r := r - h,
                      q := q / 4; h := p + q; p := p / 2]]] // Timing
```

Method is complete!

```
Out[37]= {0.593 Second, p^2 q[0] == q (p[0]^2 + q[0] (-r + r[0]))}
```

Fermat's Factoring Problem, by D. E. Knuth

```
(*
 u:=2*R+1; v:=1; r:=R*R-N;
 WHILE[r≠0,
      IF[r>0,
       r:=r-v; v:=v+2,
       r:=r+u; u:=u+2
       ]]
*)
```

```
In[56]:= Aligator[WHILE[r ≠ 0,
                  IF[r > 0,
                   r := r - v; v := v + 2,
                   r := r + u; u := u + 2
                   ]]] // Timing
```

Method is complete!

```
Out[56]= {0.109 Second, 4 r + 2 u + v^2 + u[0]^2 + 2 v[0] == u^2 + 2 v + 4 r[0] + 2 u[0] + v[0]^2}
```

```
In[57]:= Aligator[WHILE[r ≠ 0,
                  IF[r > 0,
                   r := r - v; v := v + 2,
                   r := r + u; u := u + 2
                   ], IniVal → {u := 2 * R + 1; v := 1; r := R * R - N}] // Timing
```

Method is complete!

```
Out[57]= {0.109 Second, 4 N + 4 r + 2 u + v^2 == u^2 + 2 v}
```

Hardware Integer Division Program, by Z. Manna

```
(*2 nd While *)
(*
 WHILE[(p≠1),
    ds:=ds/2; p:=p/2;
    IF[(r≥ds), r:=r-ds; q:=q+p]]
*)
```

```
In[39]:= Aligator[WHILE[(p ≠ 1),
           ds := ds / 2; p := p / 2;
             IF[(r ≥ ds), r := r - ds; q := q + p]]] // Timing
```

```
Method is complete!
```

```
Out[39]= {0.25 Second,
          ds p[0] == p ds[0] && q ds[0] + r p[0] == ds[0] q[0] + p[0] r[0] && ds q + p r == ds q[0] + p r[0]}
```

Hardware Style Division, by Z. Manna (version 2)

■ 1st Loop

```
(* 1 st Loop *)
(* WHILE[a≥b, b:=2*b;c:=2*c]*)
```

```
In[40]:= Aligator[WHILE[a ≥ b, b := 2 * b; c := 2 * c]] // Timing
```

```
Method is complete!
```

```
Out[40]= {0.078 Second, c b[0] == b c[0]}
```

■ 2nd Loop

```
(* 2 nd Loop *)
(*WHILE[…,IF[…,a:=a-b;d:=d+c];
  b:=b/2;c:=c/2]*)
```

```
In[41]:= Aligator[WHILE[…, IF[…, a := a - b; d := d + c];
            b := b / 2; c := c / 2]] // Timing
```

```
Method is complete!
```

```
Out[41]= {0.25 Second,
          b c[0] == c b[0] && d b[0] + a c[0] == a[0] c[0] + b[0] d[0] && a c + b d == c a[0] + b d[0]}
```

Binary Division, by A. Kaldewaij

```
(*2 nd While *)
(*
 WHILE[(b≠B),
       q:=2*q;b:=b/2;
       IF[(r≥b), q:=q+1;r:=r-b]
 ]
*)
```

```
In[42]:= Aligator[WHILE[(b ≠ B),
             q := 2 * q; b := b / 2;
          IF[(r ≥ b), q := q + 1; r := r - b]
         ]] // Timing
```

```
Method is complete!
```

```
Out[42]= {0.219 Second, b q + r == b[0] q[0] + r[0]}
```

Extended Euclid's Algorithm, by D. E. Knuth

```
(*
 a:=x;b:=y;p:=1; q:=0;r:=0;s:=1;
 WHILE[a≠b,
      IF[a>b,
           a:=a-b;p:=p-q;r:=r-s,
           b:=b-a;q:=q-p;s:=s-r
       ]
 ]
*)
```

```
In[43]:= Aligator[WHILE[a ≠ b,
            IF[a > b,
                 a := a - b; p := p - q; r := r - s,
                 b := b - a; q := q - p; s := s - r
             ]
         ]] // Timing
```

```
Method is complete!
```

```
Out[43]= {3.187 Second, a q + b[0] p[0] == b p + a[0] q[0] &&
          a s + b[0] r[0] == b r + a[0] s[0] && p s + q[0] r[0] == q r + p[0] s[0] &&
          r a[0] q[0] + p b[0] r[0] + a p[0] s[0] == r b[0] p[0] + a q[0] r[0] + p a[0] s[0] &&
          s a[0] q[0] + q b[0] r[0] + b p[0] s[0] == s b[0] p[0] + b q[0] r[0] + q a[0] s[0]}
```

```
In[44]:= Aligator[WHILE[a ≠ b,
                IF[a > b,
                     a := a - b; p := p - q; r := r - s,
                     b := b - a; q := q - p; s := s - r
                 ]],
           IniVal → {a := x; b := y; p := 1; q := 0; r := 0; s := 1}] // Timing
```

```
Method is complete!
```

```
Out[44]= {3.094 Second, a q + y == b p && a s == b r + x && p s == 1 + q r && a == p x + r y && b == q x + s y}
```

Binary Division (with 4 branches), by E. Rodriguez Carbonell

```
(*
 a:=x; b:=y;p:=1;q:=0;
 WHILE[((a≠0) ∧ (b≠0)),
      IF[(Mod[a,2]=0) ∧ (Mod[b,2]=0), a:=a/2;b:=b/2;p:=4*p,
           IF[(Mod[a,2]=1) ∧ (Mod[b,2]=0), a:=a-1;q:=q+b*p,
                IF[(Mod[a,2]=0) ∧ (Mod[b,2]=1),
                     b:=b-1;q:=q+a*p,
                     q:=q+(a+b-1)*p;a:=a-1;b:=b-1;q:=q+(a+b-1)*p ]]]]
*)
```

```
In[58]:= Aligator[WHILE[((a ≠ 0) ∧ (b ≠ 0)),
                IF[(Mod[a, 2] = 0) ∧ (Mod[b, 2] = 0), a := a / 2; b := b / 2; p := 4 * p,
                     IF[(Mod[a, 2] = 1) ∧ (Mod[b, 2] = 0), a := a - 1; q := q + b * p,
                          IF[(Mod[a, 2] = 0) ∧ (Mod[b, 2] = 1),
                               b := b - 1; q := q + a * p,
                               q := q + (a + b - 1) * p; a := a - 1; b := b - 1 ]]]]] // Timing
```

```
Method is complete!
```

```
Out[58]= {0.313 Second, a b p + q == a[0] b[0] p[0] + q[0]}
```

```
In[59]:= Aligator[WHILE[((a ≠ 0) ∧ (b ≠ 0)),
                IF[(Mod[a, 2] = 0) ∧ (Mod[b, 2] = 0), a := a / 2; b := b / 2; p := 4 * p,
                     IF[(Mod[a, 2] = 1) ∧ (Mod[b, 2] = 0), a := a - 1; q := q + b * p,
                          IF[(Mod[a, 2] = 0) ∧ (Mod[b, 2] = 1),
                               b := b - 1; q := q + a * p,
                               q := q + (a + b - 1) * p; a := a - 1; b := b - 1 ]]]],
           IniVal → {a := x; b := y; p := 1; q := 0}] // Timing
```

```
Method is complete!
```

```
Out[59]= {0.297 Second, a b p + q == x y}
```

Large Number's Factorization, by D. E. Knuth

```
(*
 WHILE [((s≥d)∧(r≠0)),
   IF [(2*r-rp+q<0), t:=r; r:=2*r-rp+q+d+2; rp:=t; q:=q+4;d:=d+2,
           IF [((2*r-rp+q≥0) ∧ (2*r-rp+q<d+2)), t:=r;r:=2*r-rp+q;rp:=t;d:=d+2,
                IF [((2*r-rp+q≥0)∧(2*r-rp+q≥d+2) ∧(2*r-rp+q<2*d+4)),
     t:=r; r:=2*r-rp+q-d-2; rp:=t; q:=q-4; d:=d+2,
                                t:=r; r:=2*r-rp+q-2*d-4; rp:=t; q:=q-8;d:=d+2 ]]]]
*)
```

In[60]:=

```
Aligator[WHILE[((s ≥ d) ∧ (r ≠ 0)),
    IF[(2 * r - rp + q < 0), t := r; r := 2 * r - rp + q + d + 2; rp := t; q := q + 4; d := d + 2,
            IF[((2 * r - rp + q ≥ 0) ∧ (2 * r - rp + q < d + 2)), t := r; r := 2 * r - rp + q; rp :=
                 IF[((2 * r - rp + q ≥ 0) ∧ (2 * r - rp + q ≥ d + 2) ∧ (2 * r - rp + q < 2 * d + 4)),
       t := r; r := 2 * r - rp + q - d - 2; rp := t; q := q - 4; d := d + 2,
                            t := r; r := 2 * r - rp + q - 2 * d - 4; rp := t; q := q - 8; d := d
  Timing
```

```
Method is complete!
```

Out[60]=

```
{0.906 Second,
 2 d (q + 2 r - 2 rp) + d[0]^2 q[0] + 8 r[0] + 4 d[0] rp[0] == d^2 q + 8 r + 2 d[0] (q[0] + 2 r[0])}
```