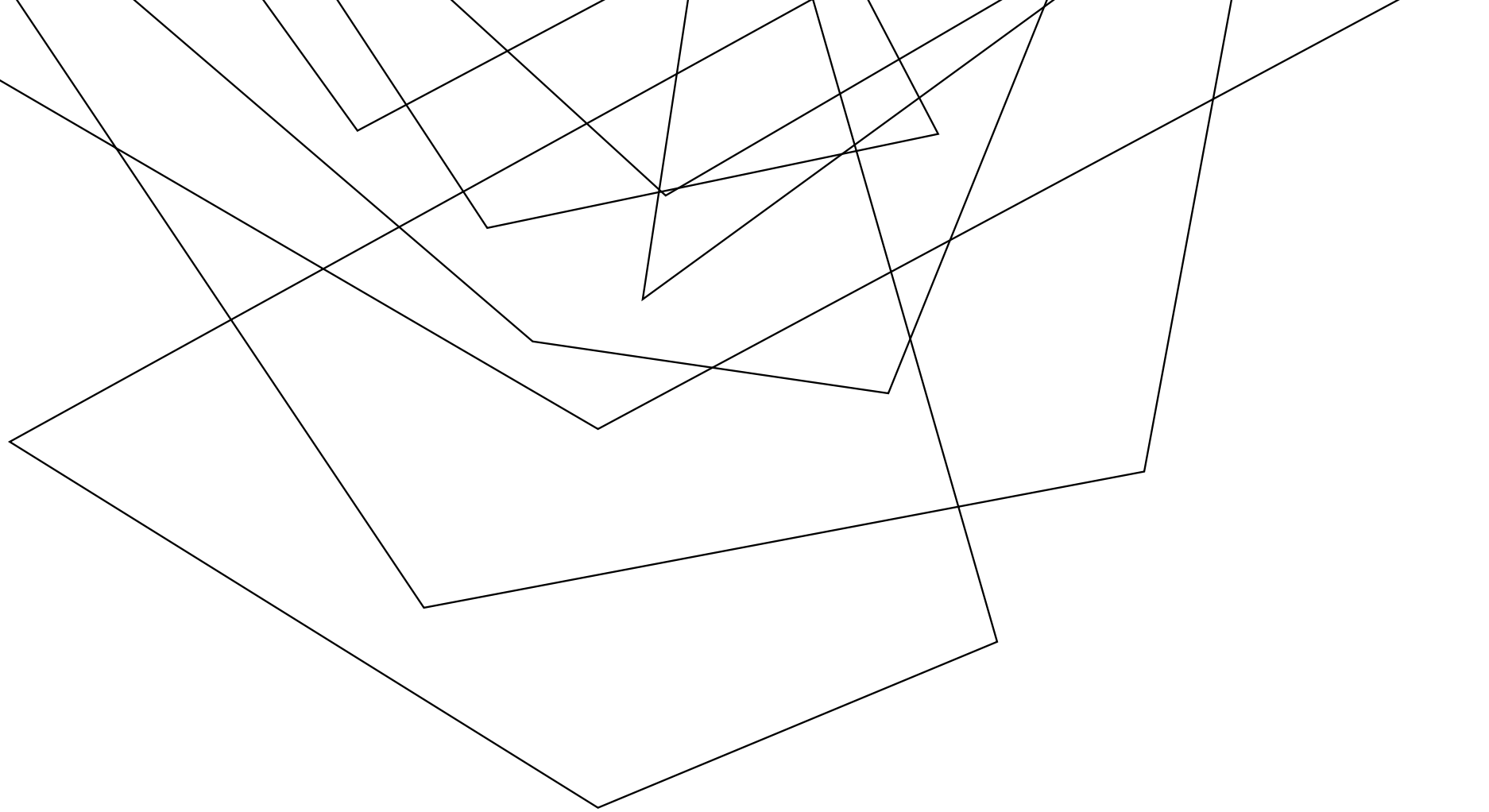


FUNKJSONER

Mick Williams

HVA ER EN FUNKSJON?

INPUT OG OUTPUT

KALL

KAN OGSÅ VÆRE EN VERDI

```
2 def fact(n):
3     if n == 0:
4         return 1
5     else:
6         return n * fact(n-1)
```

```
8 fact = (lambda n : 1 if n == 0 else n * fact(n-1))
```

REKURSJON UTEN DEFINISJONER

```
10 (lambda x : x + 3)(5)
```

```
12 (lambda proc : proc(3))(
13     (lambda num : num*2))
```

```
15 (lambda proc : (proc proc n))(
16     (lambda self n : 1 if n == 0 else n * self(self, n-1)))
```

```
18 (lambda fact :
19  fact(5))(
20      (lambda n :
21       (lambda proc : (proc proc n))(
22           (lambda self n : 1 if n == 0 else n * self(self, n-1))))
```

IF SOM EN FUNKSJON

```
26 # T
27 (lambda x, y : x)
28
29 # F
30 (lambda x, y : y)
31
32 # f_not
33 (lambda boolean : boolean(F, T))
34
```

```
35 # f_and
36 (lambda bool1, bool2 : bool1(bool2, bool1))
37
38 # f_or
39 (lambda bool1, bool2 : bool1(bool1, bool2))
40
41 # f_xor
42 (lambda bool1, bool2 : bool1(f_not(bool2), bool2))
```

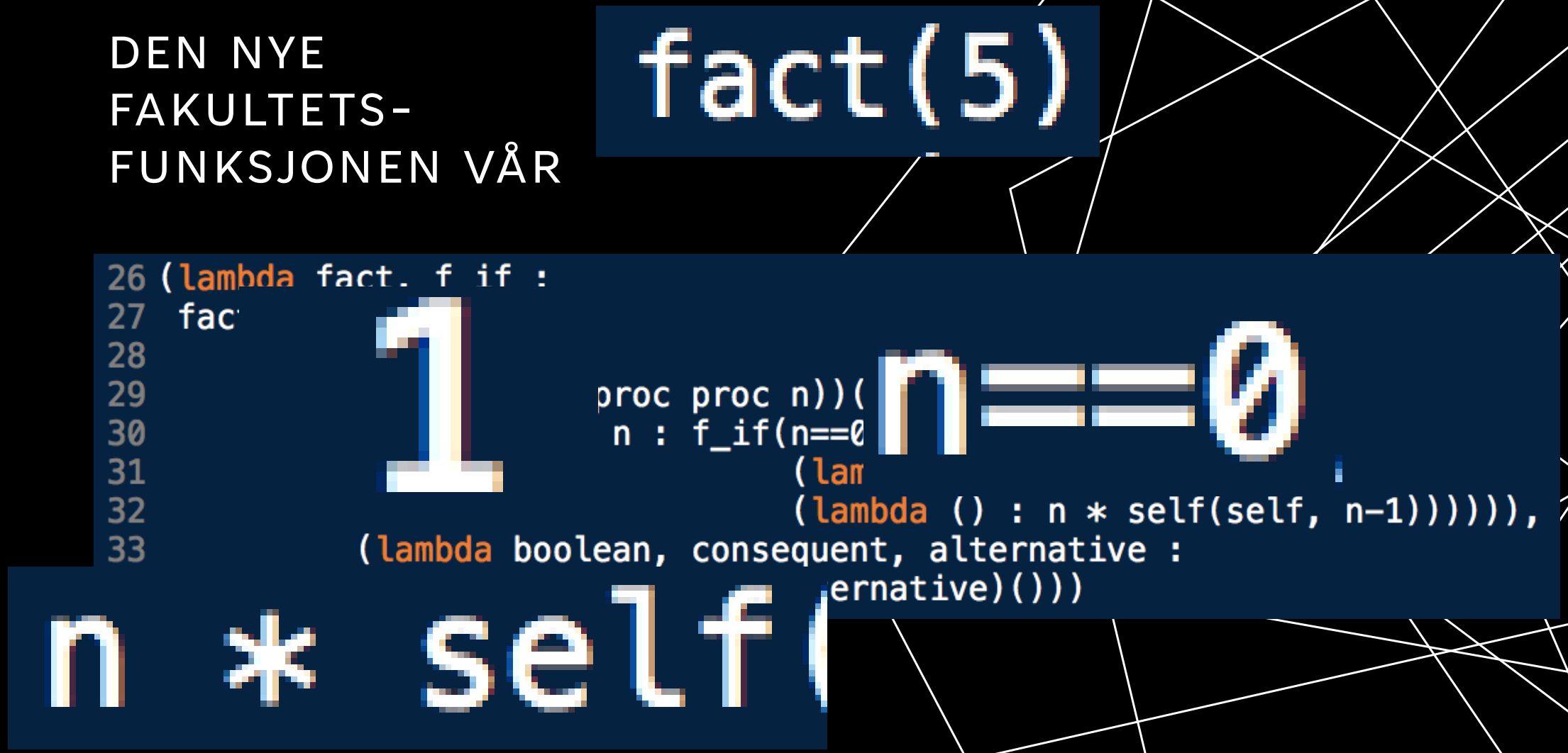
IF SOM EN FUNKSJON

```
44 # f_if
45 (lambda boolean, consequent, alternative :
46  boolean(consequent, alternative)())
```

DEN NYE FAKULTETS-FUNKSJONEN VÅR

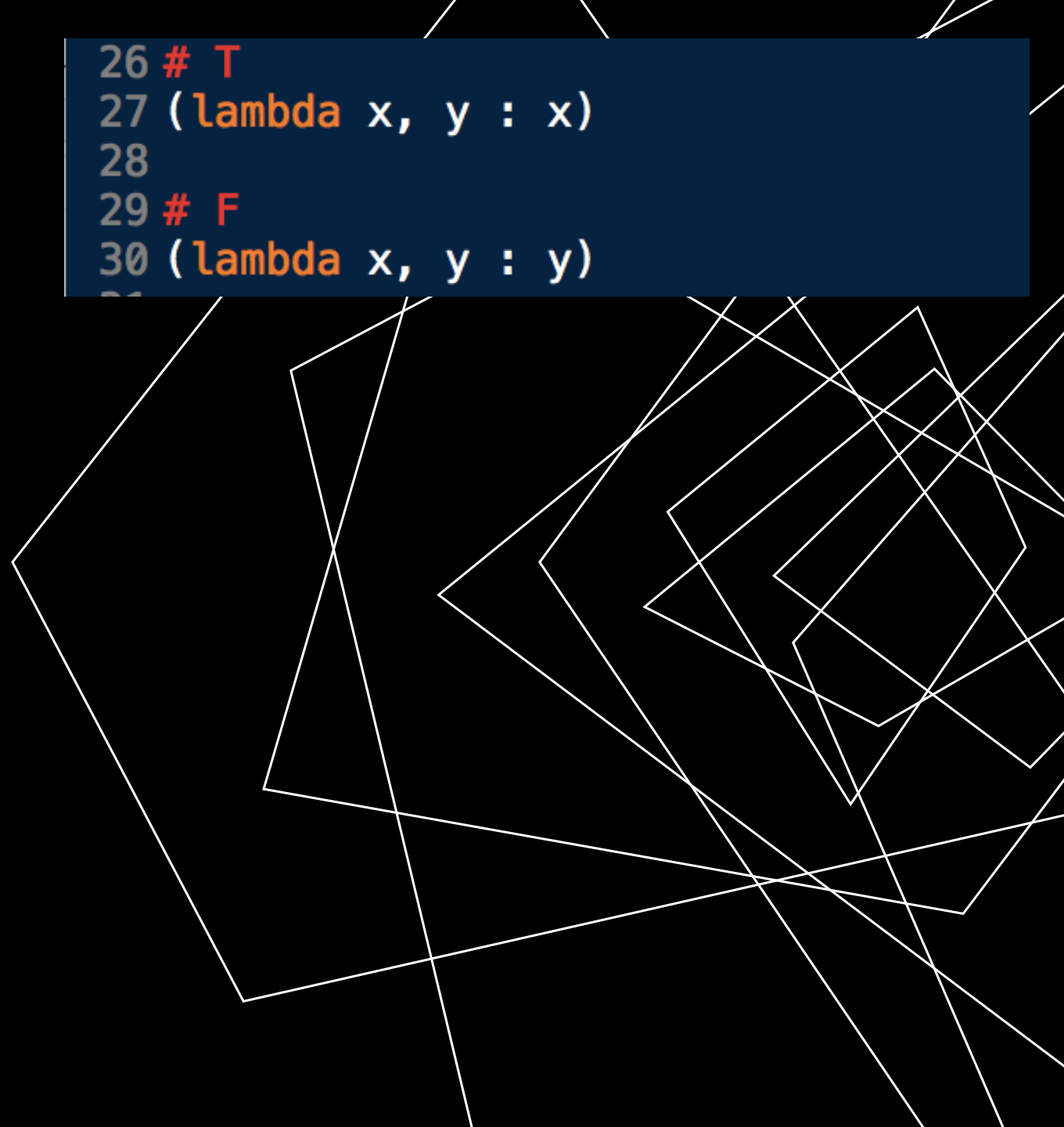
```
26 (lambda fact, f_if :
27  fact(5))(
28      (lambda n :
29       (lambda proc : (proc proc n))(
30           (lambda self n : f_if(n==0,
31                                (lambda () : 1),
32                                (lambda () : n * self(self, n-1)))))),
33           (lambda boolean, consequent, alternative :
34            boolean(consequent, alternative)()))
```

DEN NYE FAKULTETS-FUNKSJONEN VÅR



TALL SOM FUNKSJONER

```
26 # T
27 (lambda x, y : x)
28
29 # F
30 (lambda x, y : y)
```



TALL SOM FUNKSJONER

```
26 # T
27 (lambda x, y : x)
28
29 # F
30 (lambda x, y : y)
```

```
65 # make_node
66 (lambda head, tail : (lambda select : select(head, tail)))
67
68 # get_head
69 (lambda node : node((lambda head, tail : head)))
70
71 # get_tail
72 (lambda node : node((lambda head, tail : tail)))
73
```

TALL SOM FUNKSJONER

```
26 # T
27 (lambda x, y : x)
28
29 # F
30 (lambda x, y : y)
```

```
76 # make_pair
77 (lambda head, tail : make_node(make_node(F, head), tail))
78
79 # get_meta_head
80 (lambda node : node((lambda head, tail : head)))
81
82 # get_head
83 (lambda node : get_meta_head(node)((lambda x, y : y)))
84
85 # nil
86 make_node(make_node(T, T), T)
87
88 # is_nil
89 (lambda node : get_meta_head(get_meta_head(node)))
```

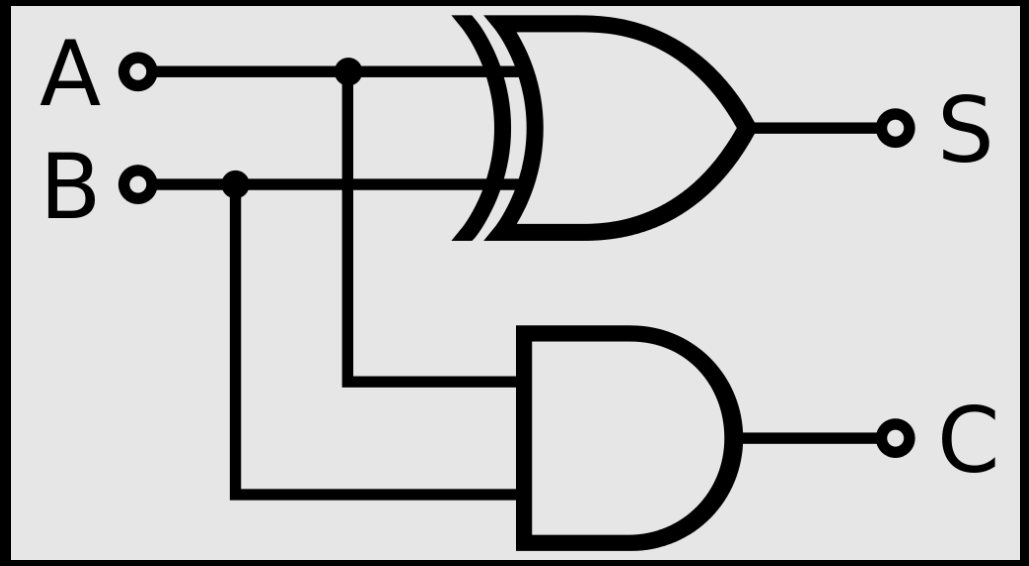
TALL SOM FUNKSJONER

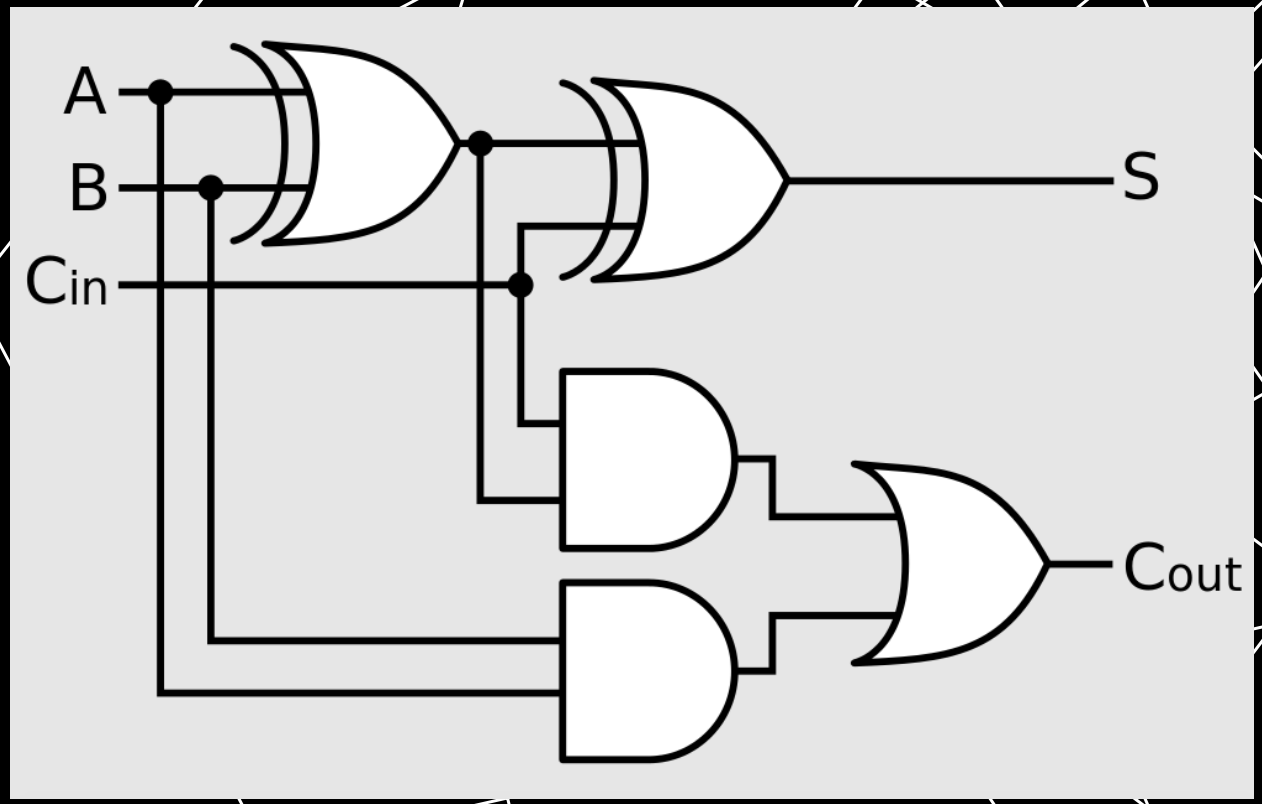
```
93 make_pair(T, make_pair(F, make_pair(F, make_pair(T, nil))))
```

LIKHET

```
132 # f_equal
133 (lambda num1, num2 :
134  (lambda proc :
135   proc(proc, num1, num2))(
136  (lambda self, num1, num2 :
137   f_if(is_nil(num1),
138        (lambda : T),
139        (lambda :
140         f_and(f_not(f_xor(get_head(num1), get_head(num2))),
141               self(self, get_tail(num1), get_tail(num2)))))))))
```

ADDERS

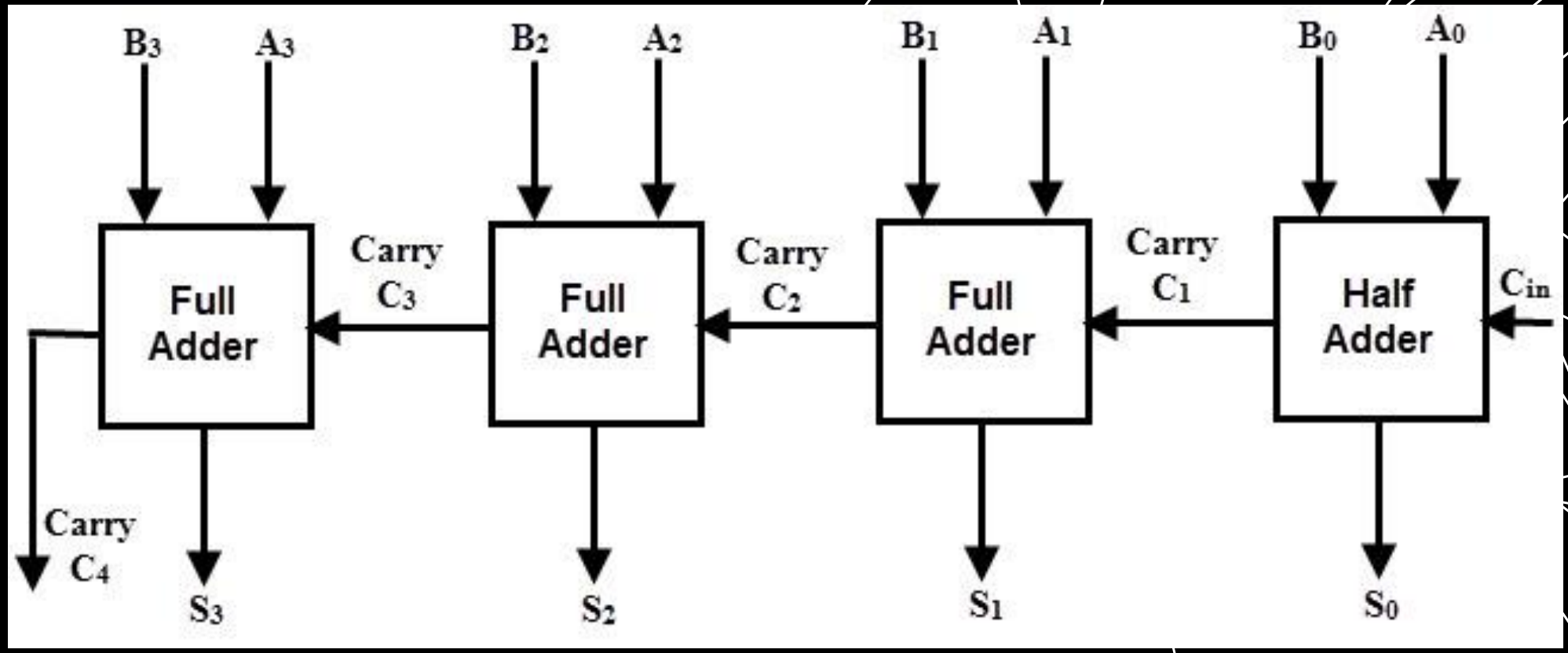




ADDERS

```
117 # full_adder
118 (lambda bit1, bit2, carry :
119  make_pair(f_xor(f_xor(bit1, bit2), carry),
120            f_or(f_and(f_xor(bit1, bit2), carry),
121                 f_and(bit1, bit2))))
122
123 # half_adder
124 (lambda bit1, bit2 : full_adder(bit1, bit2, F))
125
126 # adder_out
127 (lambda adder : get_head(adder))
```

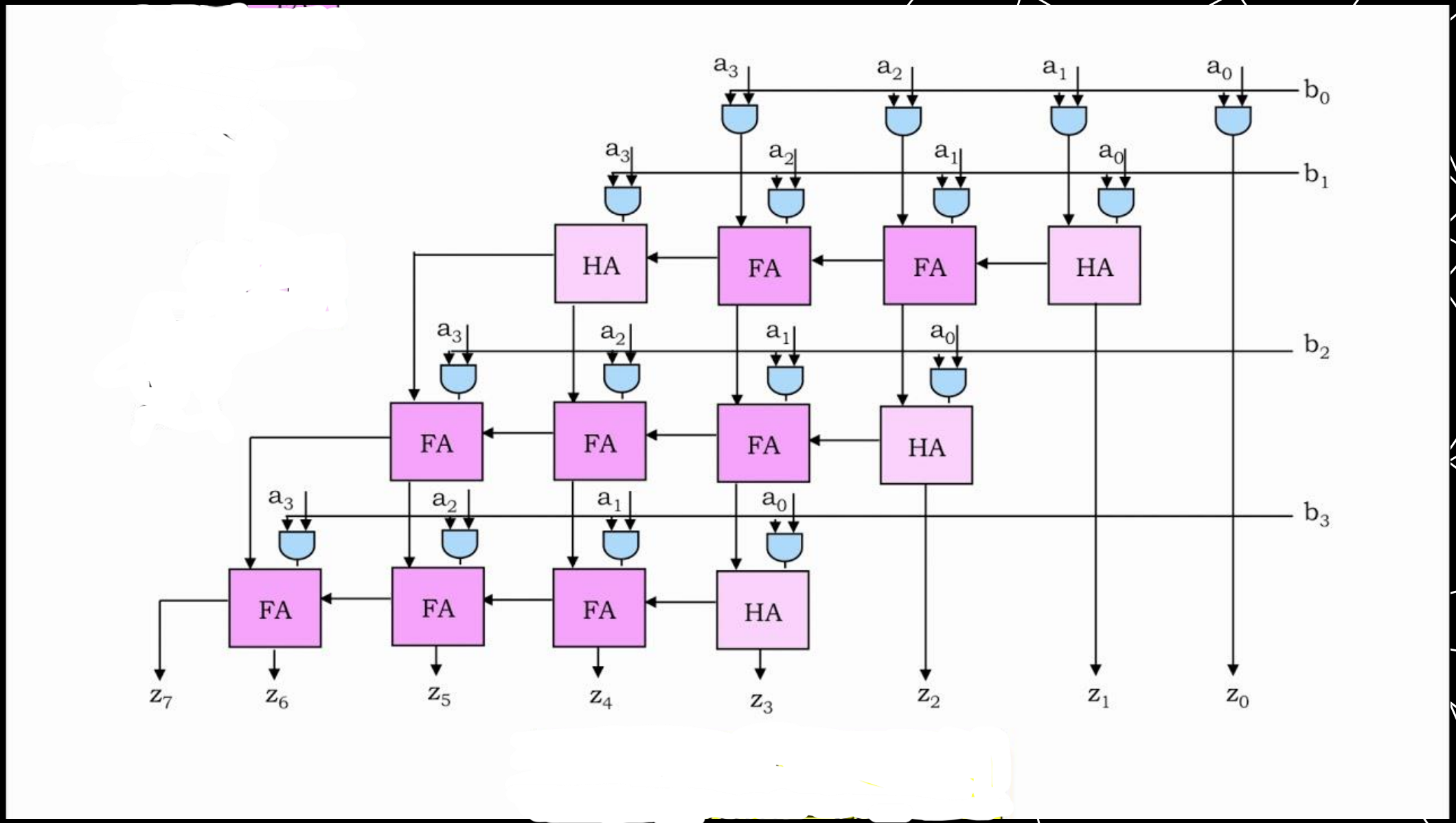
ADDISJON


B3
A3
B2
A2
B1
A1
B0
A0
Full Adder
Full Adder
Full Adder
Half Adder
Carry C3
Carry C2
Carry C1
Cin
Carry C4
S3
S2
S1
S0

ADDISJON

```
143 # f_add
144 (lambda m, n :
145  (lambda proc : get_tail(proc(proc, m, n)))(
146  (lambda self, m, n :
147   f_if(is_nil(get_tail(m)),
148        (lambda :
149         (lambda ha :
150          make_pair(adder_carry(ha), make_pair(adder_out(ha), nil)))(
151         half_adder(get_head(m), get_head(n)))),
152        (lambda :
153         (lambda next_adder :
154          (lambda fa :
155           make_pair(adder_carry(fa), make_pair(adder_out(fa), get_tail(next_adder))))(
156          full_adder(get_head(m), get_head(n), get_head(next_adder))))(
157         inner(get_tail(m), get_tail(n)))))))
```

MULTIPLIKASJON



MULTIPLIKASJON

```
171 # basis_layer
172 (lambda a, b0, b1 :
173  (lambda first_layer:
174   (lambda out :
175    (lambda second_layer :
176     make_pair(f_reverse(get_tail(f_reverse(get_head(second_layer)))),
177               make_pair(get_tail(second_layer), out)))(
178    (lambda proc :
179     (lambda rest :
180      (lambda first_ha :
181       make_pair(make_pair(first_ha, get_head(rest)),
182                 get_tail(rest))))(
183      half_adder(f_and(get_head(a), b1),
184                 adder_carry(get_head(get_head(rest)))))(
185      proc(proc, first_layer, get_tail(a)))(
186    (lambda self, layer, a :
187     f_if(is_nil(get_tail(get_tail(layer))),
188          (lambda :
189           (lambda ha :
190            make_pair(make_pair(ha, nil),
191                      adder_out(ha)))(
192           half_adder(get_head(layer),
193                      f_and(get_head(a), b1)))),
194          (lambda prev_adder :
195           make_pair(make_pair(full_adder(get_head(layer),
196                                          f_and(get_head(a), b1),
197                                          adder_carry(get_head(get_head(prev_adder)))),
198                               get_head(prev_adder)),
199                     get_tail(prev_adder)))(
200          self(self, get_tail(layer), get_tail(a))))))))))(
201   make_pair(get_head(f_reverse(first_layer)), nil)))(
202  f_reduce((lambda x, y: make_pair(f_and(x, b0), y)), nil, a)))
```

MULTIPLIKASJON

```
# basis_layer
(lambda a, b0, b1 :
 (lambda first_layer:
  (lambda out :
   (lambda second_layer :
    make_pair(f_reverse(get_tail(f_reve
              make_pair(get_tail(second
   (lambda proc :
    (lambda rest :
     (lambda first_ha :
      make_pair(make_pair(first_ha, g
                get_tail(rest))))(
     half_adder(f_and(get_head(a), b
                adder_carry(get_head
     proc(proc, first_layer, get_ta
   (lambda self, layer, a :
    f_if(is_nil(get_tail(get_tail(
         (lambda :
          (lambda ha :
           make_pair(make_pair(ha
                     adder_out(ha
           half_adder(get_head(la
                      f_and(get_h
         (lambda prev_adder :
          make_pair(make_pair(f
                    get_tail(prev_a
         self(self, get_tail(layer), get_ta
  make_pair(get_head(f_reverse(first_layer)), nil)))
 f_reduce((lambda x, y: make_pair(f_and(x, b0), y)), nil, a...
```

```
# construct_layer
(lambda prev_layer, a, B :
 (lambda out :
  (lambda inner :
   (lambda layer:
    (lambda first_adder:
     make_pair(make_pair(first_adder layer),
               adder_out(out)))(
     full_adder(adder_carry(get_head(prev_layer)),
                f_and(get_head(a), B),
                adder_carry(get_head(layer)))))(
    inner(prev_layer, get_tail(a))))(
   (lambda prev_layer, a :
    (lambda proc :
     proc(proc, prev_layer, a))(
    (lambda inner, prev_layer, a:
     f_if(is_nil(get_tail(get_tail(prev_layer))),
          (lambda :
           (lambda this_adder:
            make_pair(this_adder, nil))(
            full_adder(adder_out(get_head(prev_layer)),
                       f_and(get_head(a), B)),
                       adder_carry(out))),
          (lambda :
           (lambda layer_tail :
            (lambda this_adder:
             make_pair(this_adder, layer_tail))(
             full_adder(adder_out(get_head(prev_layer)),
                        f_and(get_head(a), B),
                        adder_carry(get_head(layer_tail)))))(
            inner(get_tail(prev_layer), get_tail(a))))))))(
    half_adder(adder_out(get_head(f_reverse(prev_layer))),
               f_and(get_head(f_reverse(a)), B)))))
```

MULTIPLIKASJON

```
# construct_layer
(lambda prev_layer
 (lambda out
```

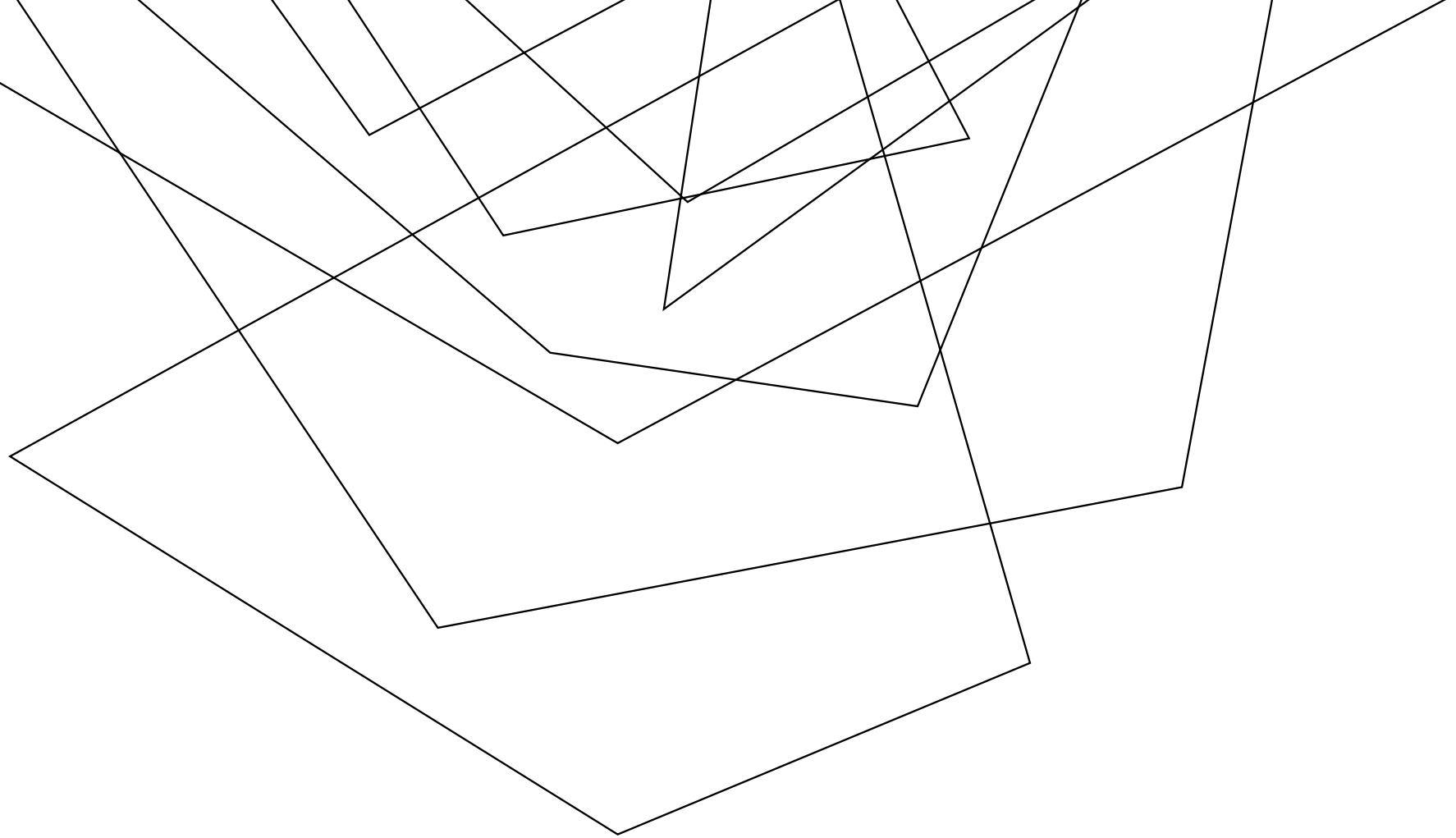
```
# f-multiply
(lambda a, b :
 (lambda inner :
  (lambda layers :
   make_pair(adder_carry(get_head(get_head(layers))),
             f_reduce((lambda x, y : make_pair(adder_out(x), y)),
                      get_tail(layers),
                      get_head(layers)))(
                                                          )),
  inner(b)))(
 (lambda b :
  (lambda proc :
   proc(proc, b))(
   (lambda inner, b:
    f_if(is_nil(get_tail(get_tail(b))),
         (lambda :
          basis_layer(a, get_head(get_tail(b)), get_head(b))),
         (lambda :
          (lambda prev_layer :
           (lambda this_layer :
            make_pair(get_head(this_layer),
                      make_pair(get_tail(this_layer),
                                get_tail(prev_layer))))(
            construct_layer(get_head(prev_layer), a, get_head(b))))(
           inner(inner, get_tail(b)))))))))
 make_pair(get_head(f_reverse(first_layer)), nil)))
f_reduce((lambda x, y: make_pair(f_and(x, b0), y)), nil,
```

DET GAMLE PROGRAMMET

```
 9 fact = (lambda n : 1 if n == 0 else n * fact(n-1))
10 fact(5)
```

DET NYE PROGRAMMET

```
# AUX
(lambda f_reduce, f_reverse :
 # LOGIC
 (lambda T, F, f_not, f_and, f_or, f_if :
  # MORE LOGIC
  (lambda f_xor :
   # PAIRS
   (lambda make_node, get_meta_head, get_tail :
    # MORE PAIRS
    (lambda nil, is_nil, make_pair, get_head :
     # ADDERS
     (lambda full_adder, adder_out, adder_carry :
      # ARITHMETIC (+ half-adder)
      (lambda half_adder, f_equal, f_one, f_add :
       # MULTIPLICATION
       (lambda basis_layer, construct_layer :
        (lambda f_multiply :
         # FACTORIAL
         (lambda fact :
          # CODE
          print(fact(5)))(
         (lambda n :
          (lambda uno :
           (lambda proc :
            proc(proc, uno))(
           (lambda self, m:
            f_if(f_equal(m, n),
                 (lambda
                  m),
                 (
                                                uno)))))))(
        f
       (l
        m                                   et_head(layers))),
          f_reduce((lambda x, y : make_pair(adder_out(x), y)),
                   get_head(layers))))(
    inner(b)))(
   (lambda b :
    (lambda proc :
     proc(proc, b))(
    (lambda inner, b:
     f_if(is_nil(get_tail(get_tail(b))),
          (lambda :
           basis_layer(a, get_head(get_tail(b)), get_head(b))),
          (lambda :
           (lambda prev_layer :
            (lambda this_layer :
             make_pair(get_head(this_layer),
                       make_pair(get_tail(this_layer),
                                 get_tail(prev_layer))))(
            construct_layer(get_head(prev_layer), a, get_head(b))))(
           inner(inner, get_tail(b)))))))))))(
  (lambda a, b0, b1 :
   (lambda first_layer:
    (lambda out :
     (lambda second_layer :
      make_pair(f_reverse(get_tail(f_reverse(get_head(second_layer)))),
                make_pair(get_tail(second_layer), out)))(
     (lambda proc :
      (lambda rest :
       (lambda first_ha :
        make_pair(make_pair(first_ha, get_head(rest)),
                  get_tail(rest))))(
        half_adder(f_and(get_head(a), b1),
                   adder_carry(get_head(get_head(rest)))))(
       proc(proc, first_layer, get_tail(a))))(
      (lambda self, layer, a :
       f_if(is_nil(get_tail(get_tail(layer))),
            (lambda :
             (lambda ha :
              make_pair(make_pair(ha, nil),
                        adder_out(ha)))(
              half_adder(get_head(layer),
                         f_and(get_head(a), b1)))),
            (lambda prev_adder :
             make_pair(make_pair(full_adder(get_head(layer),
                                            f_and(get_head(a), b1),
                                            adder_carry(get_head(get_head(prev_adder)))),
                                 get_head(prev_adder)),
                       get_tail(prev_adder)))(
             self(self, get_tail(layer), get_tail(a))))))))))(
   make_pair(get_head(f_reverse(first_layer)), nil)))(
  f_reduce((lambda x, y: make_pair(f_and(x, b0), y)), nil, a))),
 (lambda prev_layer, a, B :
  (lambda out :
   (lambda inner :
    (lambda layer:
     (lambda first_adder:
      make_pair(make_pair(first_adder(layer),
                          adder_out(out)))(
      full_adder(adder_carry(get_head(prev_layer)),
                 f_and(get_head(a), B),
                 adder_carry(get_head(layer)))))(
     inner(prev_layer, get_tail(a))))(
    (lambda prev_layer, a :
     (lambda proc :
      proc(proc, prev_layer, a))(
      (lambda inner, prev_layer, a:
       f_if(is_nil(get_tail(get_tail(prev_layer))),
            (lambda :
             (lambda this_adder:
              make_pair(this_adder, nil))(
              full_adder(adder_out(get_head(prev_layer)),
                         f_and(get_head(a), B)),
                         adder_carry(out))),
            (lambda :
             (lambda layer_tail :
              (lambda this_adder:
               make_pair(this_adder, layer_tail))(
               full_adder(adder_out(get_head(prev_layer)),
                          f_and(get_head(a), B),
                          adder_carry(get_head(layer_tail)))))(
             inner(get_tail(prev_layer), get_tail(a))))))))))(
   half_adder(adder_out(get_head(f_reverse(prev_layer))),
              f_and(get_head(f_reverse(a)), B))))))(
 (lambda bit1, bit2 : full_adder(bit1, bit2, F)),
 (lambda num1, num2 :
  (lambda proc :
   proc(proc, num1, num2))(
  (lambda self, num1, num2 :
   f_if(is_nil(num1),
        (lambda : T),
        (lambda :
         f_and(f_not(f_xor(get_head(num1), get_head(num2))),
               self(self, get_tail(num1), get_tail(num2)))))))),
 (lambda x :
  (lambda proc : proc(proc, x))(
  (lambda self, x :
   f_if(is_nil(get_tail(x)),
        (lambda :
         make_pair(T, nil)),
        (lambda :
         make_pair(F, self(self, get_t                  )),
 (lambda m, n :
  (lambda proc : get_tail(proc(proc
  (lambda self, m, n :
   f_if(is_nil(get_tail(m)),
        (lambda :
         (lambda ha :
          ma  e_pair(adder_carry(                    out(ha), nil)))(
          ha    der(get_head(m),
        (lambda :
         (lambda next_adder :
          (lambda fa :
           make_pair(adder_carry(fa), make_pair(adder_out(fa), get_tail(next_adder))))(
          full_adder(get_head(m), get_head(n), get_head(next_adder))))(
         inner(get_tail(m), get_tail(n)))))))))(
 (lambda bit1, bit2, carry :
  make_pair(f_xor(f_xor(bit1, bit2), carry),
            f_or(f_and(f_xor(bit1, bit2), carry),
                 f_and(bit1, bit2)))),
 (lambda adder : get_head(adder)),
 (lambda adder : get_tail(adder)))(
make_node(make_node(T, T), T),
(lambda node : get_meta_head(get_meta_head(node))),
(lambda head, tail : make_node(make_node(F, head), tail)),
(lambda node : get_meta_head(node)((lambda x, y : y))))(
(lambda head, tail : (lambda select : select(head, tail))),
(lambda node : node((lambda head, tail : head))),
(lambda node : node((lambda head, tail : tail))))(
(lambda bool1, bool2 : bool1(f_not(bool2), bool2)))(
(lambda x, y : x),
(lambda x, y : y),
(lambda boolean : boolean(F, T)),
(lambda bool1, bool2 : bool1(bool2, bool1)),
(lambda bool1, bool2 : bool1(bool1, bool2)),
(lambda boolean, consequent, alternative : boolean(consequent, alternative)))(
(lambda proc, init, items :
 (lambda self : self(self, proc, init, items))(
 (lambda self, proc, init, items :
  f_if(is_nil(items),
       (lambda : init),
       (lambda : proc(get_head(items),
                      self(self, proc, init, get_tail(items))))))),
(lambda lst :
 (lambda proc :
  proc(proc, lst, nil))(
 (lambda self, old, new:
  f_if(is_nil(old),
       (lambda : new),
       (lambda : self(get_tail(old),
                      make_pair(get_head(old), new)))))))
```



SPØRSMÅL?

Takk til Thomas Marinelli Johansen og Martin Mihle Nygaard :)