5.4 Recursive algorithms

Recall factorial function
(both non-recursive and recursive definitions):

$f(n) = n! = n\,(n-1)\,\ldots\,1$, for $n \geq 0$ and $f(0) = 0! = 1$

Then $f(4) = 4! = 4 \cdot 3 \cdot 2 \cdot 1 = 24$
$f(5) = 5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$

Recursive definition of factorial function:

$f(0) = 1$

$f(n) = n\, f(n-1)$, for $n \geq 1$

Then

$f(4) = 4\, f(3) = 4 \cdot 3\, f(2) = 4 \cdot 3 \cdot 2\, f(1) = 4 \cdot 3 \cdot 2 \cdot 1 \cdot 1$

5.4 Recursive algorithms

Let's see the pseudo-codes for each of the procedures

f(n) = n! = n (n-1) … 1, for n ≥ 0 and f(0) =0! = 1

procedure *factorial* (*n*: nonnegative integer)
fact := 1
for *i:=1* **to** *n*
    *fact := fact * i*
return *fact*
{output: n!}

5.4 Recursive algorithms

Let's see the pseudo-codes for each of the procedures

execute factorial(4)

procedure *factorial* (*n*: nonnegative integer)
fact := 1
for *i:=1* **to** *n*
    *fact := fact * i*
return *fact*
{output: n!}

5.4 Recursive algorithms

Let's see the pseudo-codes for each of the procedures

execute factorial(4)

procedure *factorial* (*4*: nonnegative integer)
fact := 1
for *i:=1* **to** *n*
    *fact := fact * i*
return *fact*
{output: n!}

fact = 1

5.4 Recursive algorithms

Let's see the pseudo-codes for each of the procedures

execute factorial(4)

procedure *factorial* (*4*: nonnegative integer)
fact := 1
for *i:=1* **to** *4*
    *fact := fact * i*
return *fact*
{output: n!}

i = 1

fact = 1

5.4 Recursive algorithms

Let's see the pseudo-codes for each of the procedures

execute factorial(4)

procedure *factorial* (*4*: nonnegative integer)
fact := 1
for *i:=1* **to** *4*
    *fact := fact * i*
return *fact*
{output: n!}

i = 1

fact = 1

5.4 Recursive algorithms

Let's see the pseudo-codes for each of the procedures

execute factorial(4)

procedure *factorial* (*4*: nonnegative integer)
fact := 1
for *i:=1* **to** *4*
    *fact := 1 * 1*
return *fact*
{output: n!}

i = 1

fact = 1

5.4 Recursive algorithms

Let's see the pseudo-codes for each of the procedures

execute factorial(4)

procedure *factorial* (*4*: nonnegative integer)
fact := 1
for *i:=1* **to** *4*
    *fact := fact * i*
return *fact*
{output: n!}

i = 2

fact = 1

5.4 Recursive algorithms

Let's see the pseudo-codes for each of the procedures

execute factorial(4)

procedure *factorial* (*4*: nonnegative integer)
fact := 1
for *i:=1* **to** *4*
    *fact := fact * i*
return *fact*
{output: n!}

i = 2

fact = 1

5.4 Recursive algorithms

Let's see the pseudo-codes for each of the procedures

execute factorial(4)

procedure *factorial* (*4*: nonnegative integer)
fact := 1
for *i:=1* **to** *4*
    *fact := 1 * 2*
return *fact*
{output: n!}

i = 2

fact = 2

5.4 Recursive algorithms

Let's see the pseudo-codes for each of the procedures

execute factorial(4)

procedure *factorial* (*4*: nonnegative integer)
fact := 1
for *i:=1* **to** *4*
    *fact := fact * i*
return *fact*
{output: n!}

i = 3

fact = 2

5.4 Recursive algorithms

Let's see the pseudo-codes for each of the procedures

execute factorial(4)

procedure *factorial* (*4*: nonnegative integer)
fact := 1
for *i:=1* **to** *4*
    *fact := fact * i*
return *fact*
{output: n!}

i = 3

fact = 2

5.4 Recursive algorithms

Let's see the pseudo-codes for each of the procedures

execute factorial(4)

procedure *factorial* (*4*: nonnegative integer)
fact := 1
for *i:=1* **to** *4*
    *fact := 2 * 3*
return *fact*
{output: n!}

i = 3

fact = 6

5.4 Recursive algorithms

Let's see the pseudo-codes for each of the procedures

execute factorial(4)

procedure *factorial* (*4*: nonnegative integer)
fact := 1
for *i:=1* **to** *4*
    *fact := fact * i*
return *fact*
{output: n!}

i = 4

fact = 6

5.4 Recursive algorithms

Let's see the pseudo-codes for each of the procedures

execute factorial(4)

procedure *factorial* (*4*: nonnegative integer)
fact := 1
for *i:=1* **to** *4*
    *fact := fact * i*
return *fact*
{output: n!}

i = 4

fact = 6

5.4 Recursive algorithms

Let's see the pseudo-codes for each of the procedures

execute factorial(4)

procedure *factorial* (*4*: nonnegative integer)
fact := 1
for *i:=1* **to** *4*
    *fact := 6 * 4*
return *fact*
{output: n!}

i = 4

fact = 24

5.4 Recursive algorithms

Let's see the pseudo-codes for each of the procedures

execute factorial(4)

procedure *factorial* (*4*: nonnegative integer)
fact := 1
for *i:=1* **to** *4*
    *fact := 6 * 4*
return *fact*
{output: n!}

i = 4

fact = 24

5.4 Recursive algorithms

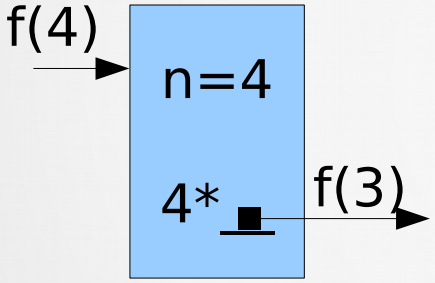
Let's see the pseudo-codes for each of the procedures

$f(0) = 1$
$f(n) = n\, f(n-1)$, for $n \geq 1$

procedure *factorial_rec* (*n*: nonnegative integer)
if *n = 0* **then return** *1*
else return *n*factorial_rec(n-1)*
{output: n!}

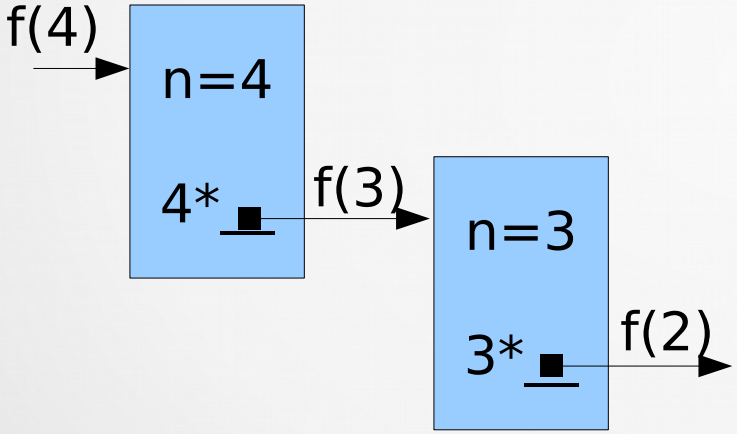
5.4 Recursive algorithms

procedure *factorial_rec* (*n*: nonnegative integer)
if *n = 0* **then return** *1*
else return *n*factorial_rec(n-1)*
{output: n!}



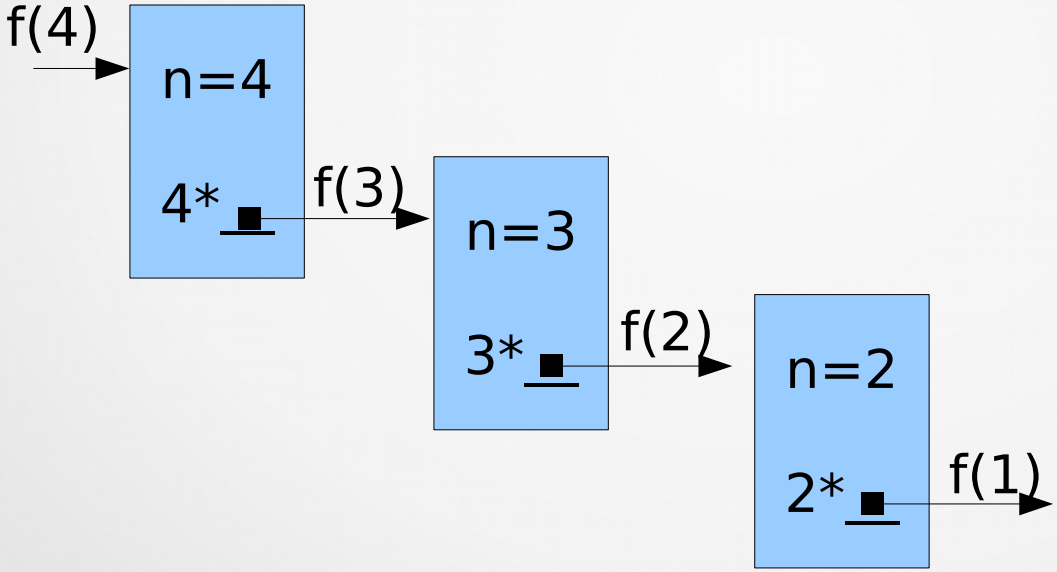
5.4 Recursive algorithms

procedure *factorial_rec* (*n*: nonnegative integer)
if *n = 0* **then return** *1*
else return *n*factorial_rec(n-1)*
{output: n!}



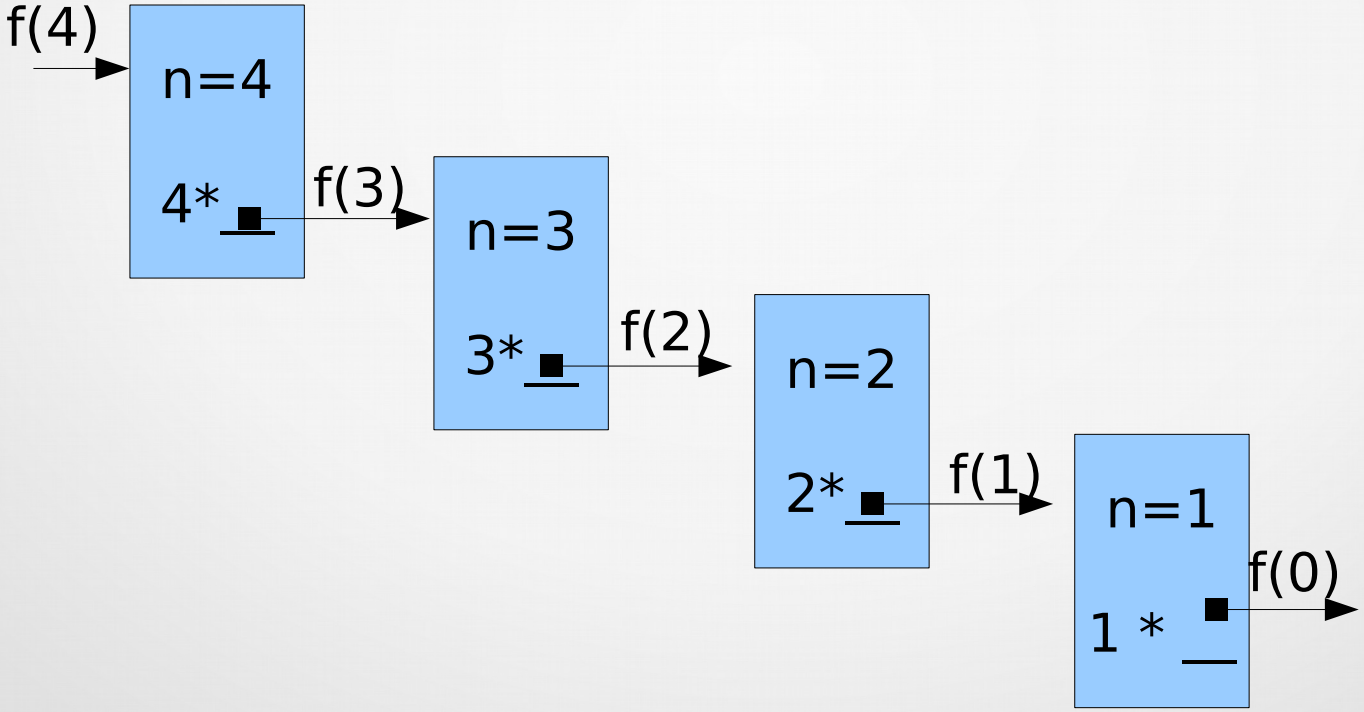
5.4 Recursive algorithms

procedure *factorial_rec* (*n*: nonnegative integer)
if *n = 0* **then return** *1*
else return *n*factorial_rec(n-1)*
{output: n!}



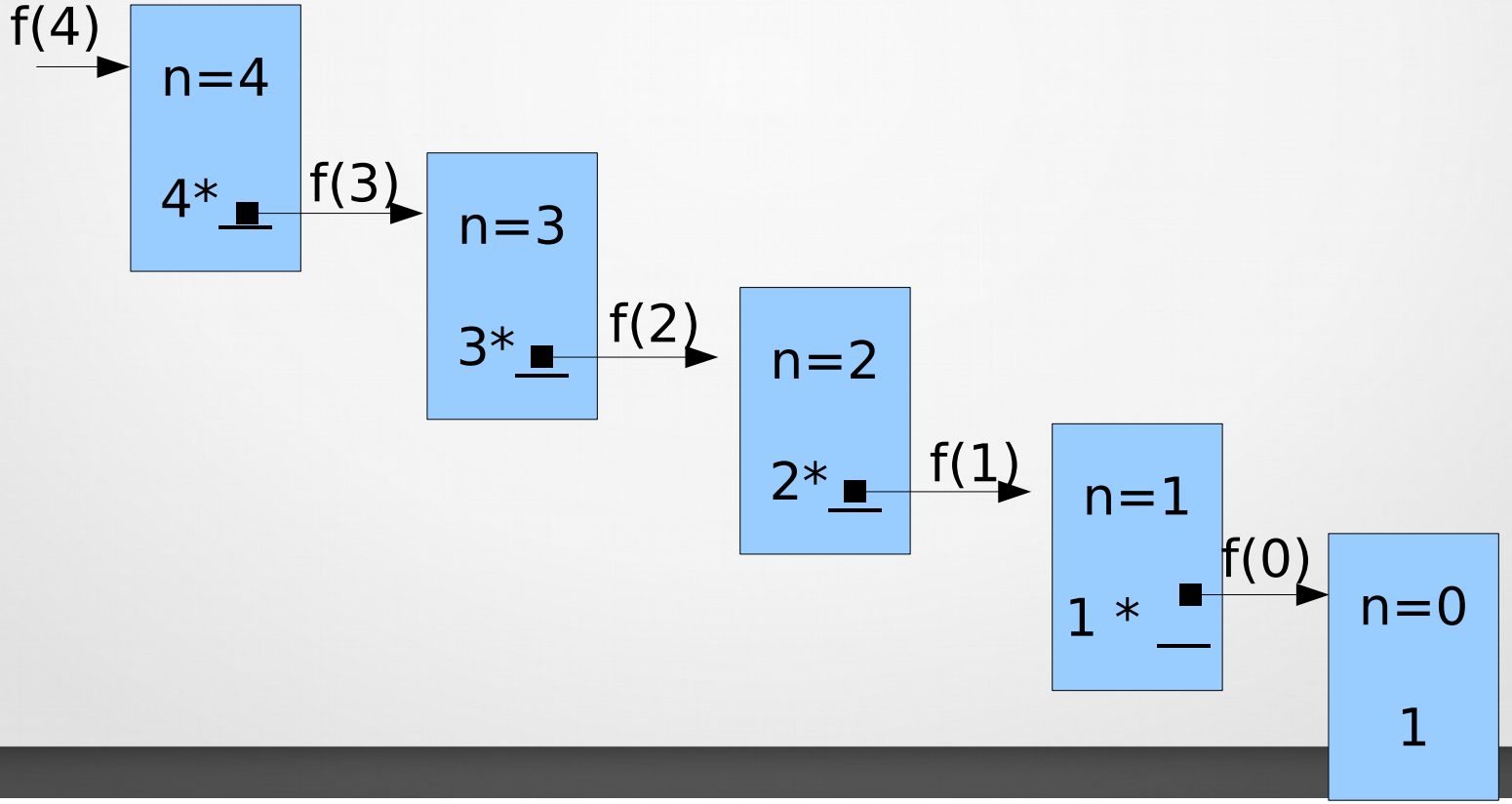
5.4 Recursive algorithms

procedure *factorial_rec* (*n*: nonnegative integer)
if *n = 0* **then return** *1*
else return *n*factorial_rec(n-1)*
{output: n!}



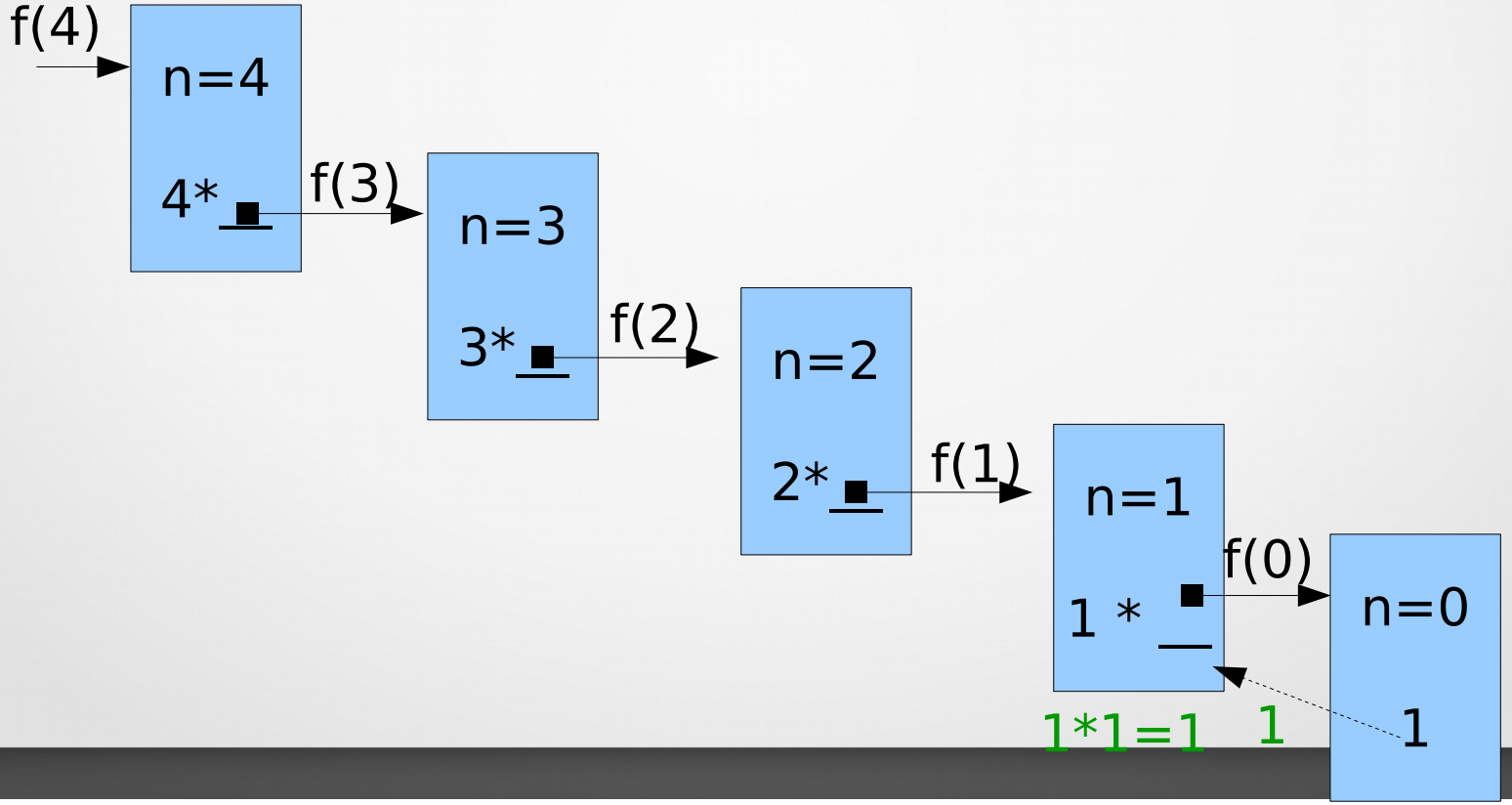
5.4 Recursive algorithms

procedure *factorial_rec* (*n*: nonnegative integer)
if *n = 0* **then return** *1*
else return *n*factorial_rec(n-1)*
{output: n!}



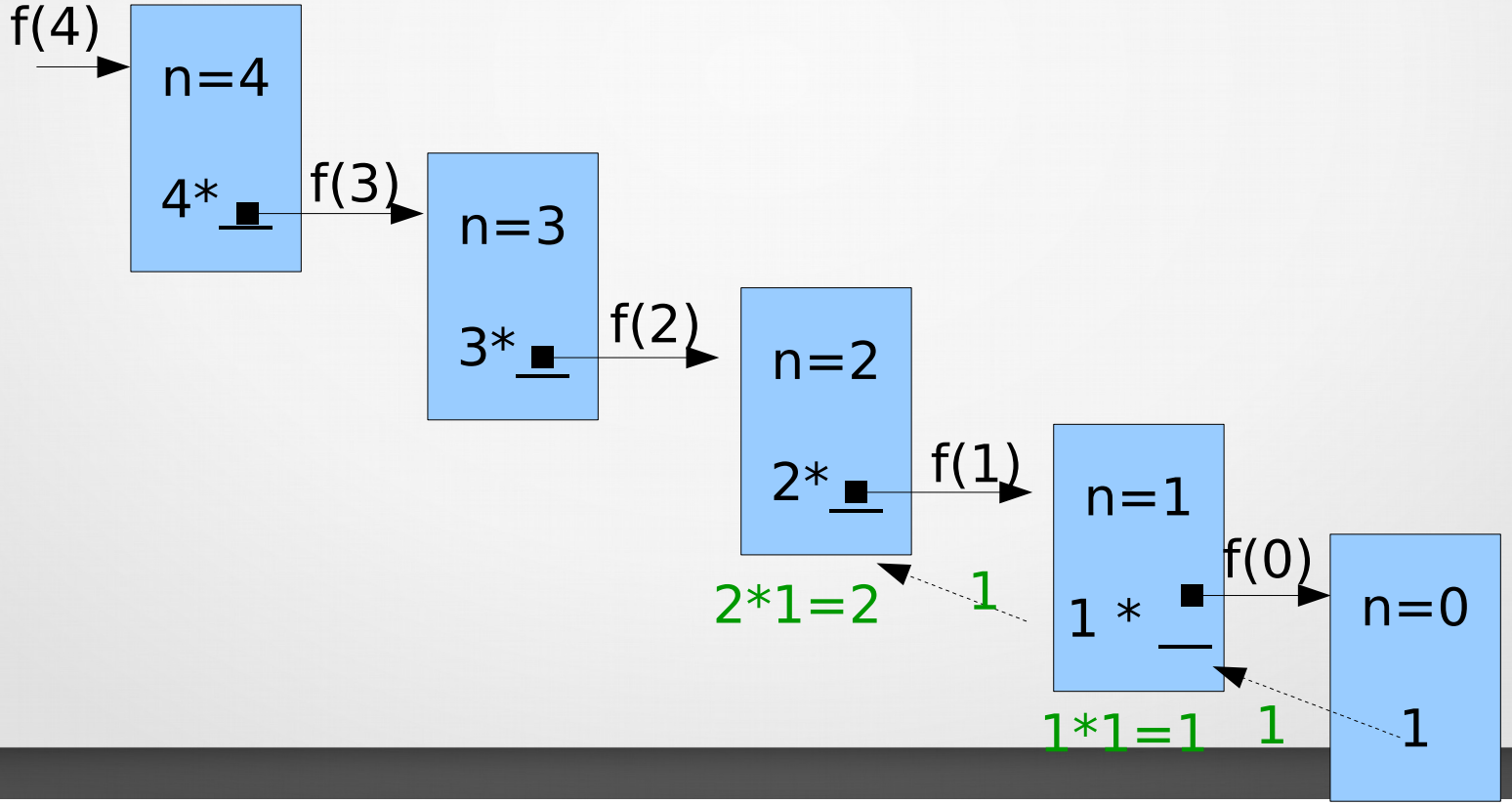
5.4 Recursive algorithms

procedure *factorial_rec* (*n*: nonnegative integer)
if *n = 0* **then return** *1*
else return *n*factorial_rec(n-1)*
{output: n!}



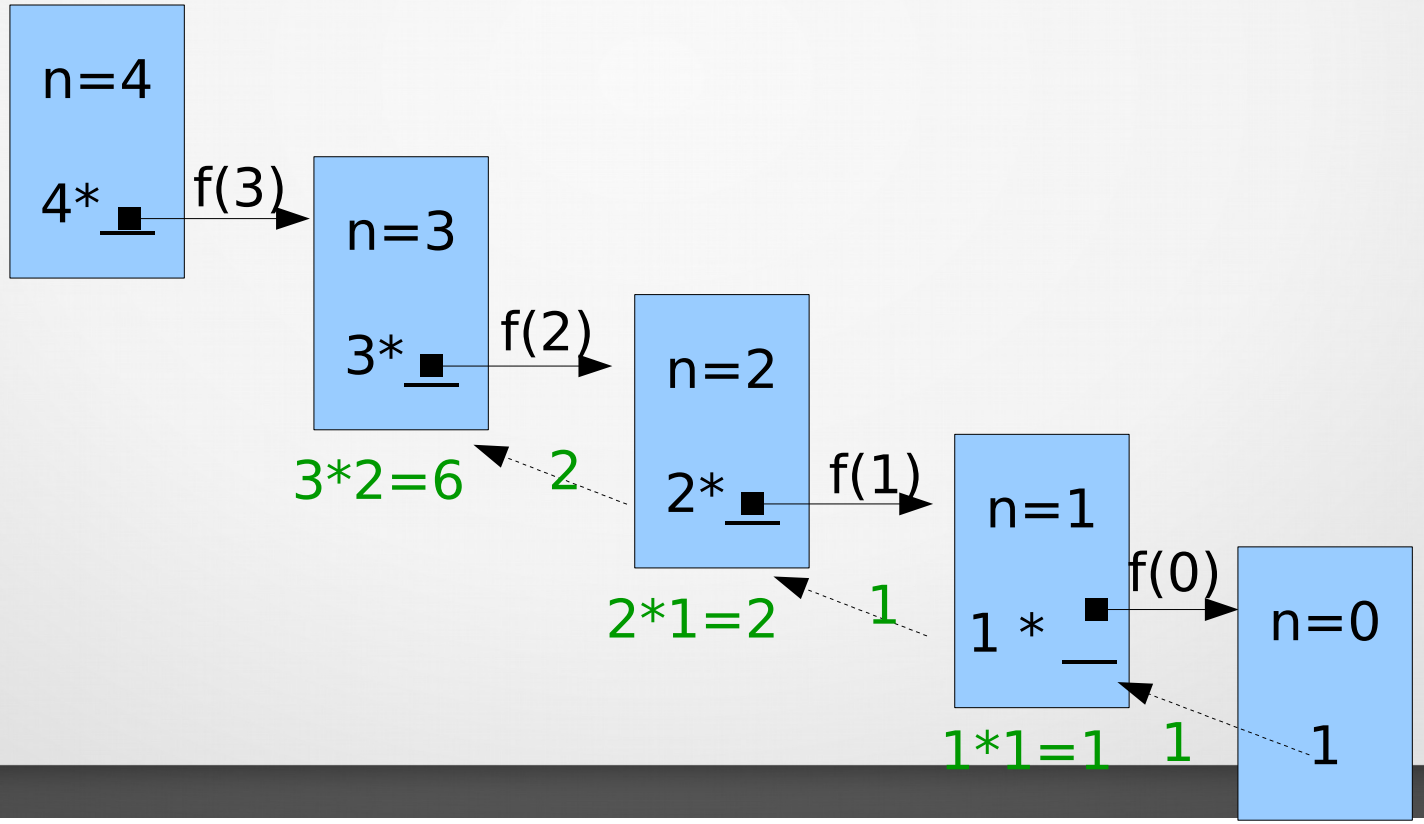
5.4 Recursive algorithms

procedure *factorial_rec* (*n*: nonnegative integer)
if *n = 0* **then return** *1*
else return *n*factorial_rec(n-1)*
{output: n!}



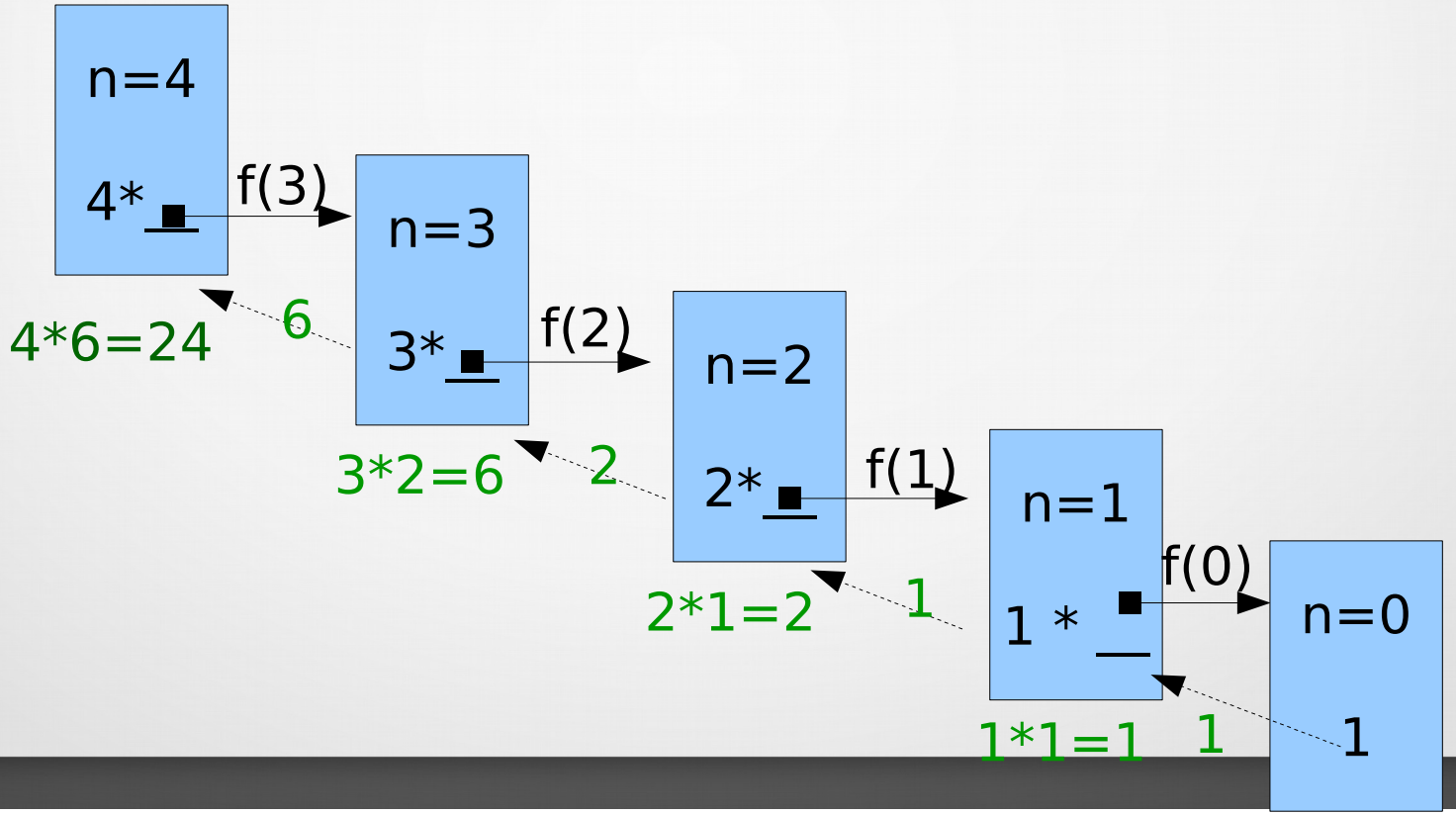
5.4 Recursive algorithms

procedure *factorial_rec* (*n*: nonnegative integer)
if *n = 0* **then return** *1*
else return *n*factorial_rec(n-1)*
{output: n!}



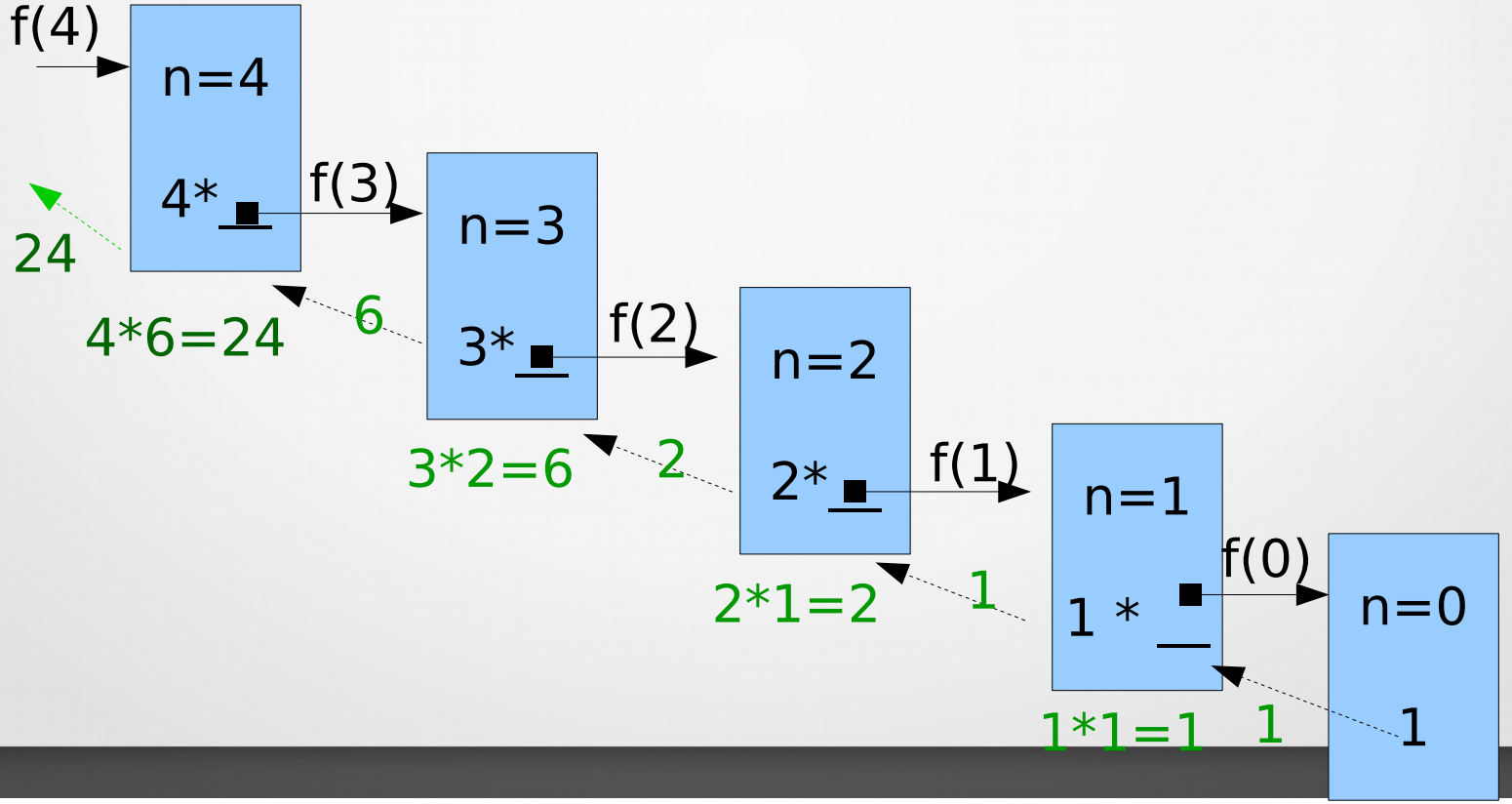
5.4 Recursive algorithms

procedure *factorial_rec* (*n*: nonnegative integer)
if *n = 0* **then return** *1*
else return *n*factorial_rec(n-1)*
{output: n!}



5.4 Recursive algorithms

procedure *factorial_rec* (*n*: nonnegative integer)
if *n = 0* **then return** *1*
else return *n*factorial_rec(n-1)*
{output: n!}



5.4 Recursive algorithms

A recursive algorithm is an algorithm that calls itself.

- has a *base case*
(the output is computed directly on an input of small size or value)

- on a larger input, the algorithm calls itself on an input of smaller size and uses the result to construct a solution to the larger input.

An algorithm's calls to itself are known as recursive calls.

5.4 Recursive algorithms

Fibonacci numbers

0 1 1 2 3 5 8 13 21 44 65 ...

f_0 f_1 f_2 f_3 f_4 f_5 …

$$f_n = f_{n-1} + f_{n-2}$$

procedure *fibonacci* (*n*: nonnegative integer)
if *n = 0* **then return** *0*
else if *n =1* **then return** *1*
else
    $f_{prev} := 1,\ f_{pprev} := 0$
    for i:=2 **to** n
        $f := f_{prev} + f_{pprev},\quad f_{prev} := f,\ f_{pprev} := f_{prev}$
return *f*
{output: n^{th} Fibonacci number}

5.4 Recursive algorithms

Fibonacci numbers

0 1 1 2 3 5 8 13 21 44 65 ...

f_0 f_1 f_2 f_3 f_4 f_5 **...**

$f_n = f_{n-1} + f_{n-2}$ call fibonacci(4)

procedure *fibonacci* (*n*: nonnegative integer)
if *n = 0* **then return** *0*
else if *n =1* **then return** *1*
else
    $f_{prev} := 1,\ f_{pprev} := 0$
    for i:=2 **to** n
        $f := f_{prev} + f_{pprev},\ f_{prev} := f,\ f_{pprev} := f_{prev}$
return *f*
{output: n^{th} Fibonacci number}

f_{pprev} :

f_{prev} :

f :

5.4 Recursive algorithms

Fibonacci numbers

0 1 1 2 3 5 8 13 21 44 65 ...

f_0 f_1 f_2 f_3 f_4 f_5 **...**

$f_n = f_{n-1} + f_{n-2}$ call fibonacci(4)

procedure *fibonacci* (*4*: nonnegative integer)
if *n = 0* **then return** *0*
else if *n =1* **then return** *1*
else
    $f_{prev} := 1,\ f_{pprev} := 0$
    for i:=2 **to** n
        $f := f_{prev} + f_{pprev},\ f_{prev} := f,\ f_{pprev} := f_{prev}$
return *f*
{output: nth Fibonacci number}

f_{pprev} :

f_{prev} :

f :

5.4 Recursive algorithms

Fibonacci numbers

0 1 1 2 3 5 8 13 21 44 65 ...

f_0 f_1 f_2 f_3 f_4 f_5 …

$f_n = f_{n-1} + f_{n-2}$ call fibonacci(4)

procedure *fibonacci* (*4*: nonnegative integer)
if *n = 0* **then return** *0*
else if *n =1* **then return** *1*
else
f_{prev}: = 1, f_{pprev} : = 0
for i:=2 **to** n
f := f_{prev} + f_{pprev}, f_{prev} := f, f_{pprev} := f_{prev}
return *f*
{output: n^{th} Fibonacci number}

f_{pprev} : 0

f_{prev} : 1

f :

5.4 Recursive algorithms

Fibonacci numbers

0 1 1 2 3 5 8 13 21 44 65 ...

f_0 f_1 f_2 f_3 f_4 f_5 …

$f_n = f_{n-1} + f_{n-2}$ call fibonacci(4)

procedure *fibonacci* (*4*: nonnegative integer)
if *n = 0* **then return** *0*
else if *n =1* **then return** *1*
else
    $f_{prev} := 1,\ f_{pprev} := 0$
    for i:=2 **to** n
        $f := f_{prev} + f_{pprev},\ f_{prev} := f,\ f_{pprev} := f_{prev}$
return *f*
{output: nth Fibonacci number}

f_{pprev} : 0

f_{prev} : 1

f :

i = 2

5.4 Recursive algorithms

Fibonacci numbers

0 1 1 2 3 5 8 13 21 44 65 ...

f_0 f_1 f_2 f_3 f_4 f_5 …

$f_n = f_{n-1} + f_{n-2}$ call fibonacci(4)

procedure *fibonacci* (*4*: nonnegative integer)
if *n = 0* **then return** *0*
else if *n =1* **then return** *1*
else
    $f_{prev} := 1,\ f_{pprev} := 0$
    for i:=2 **to** n
        $f := f_{prev} + f_{pprev},\ f_{prev} := f,\ f_{pprev} := f_{prev}$
return *f*
{output: nth Fibonacci number}

f_{pprev} : 0

f_{prev} : 1

f : 1+0 = 1

i = 2

5.4 Recursive algorithms

Fibonacci numbers

0 1 1 2 3 5 8 13 21 44 65 ...

f_0 f_1 f_2 f_3 f_4 f_5 ...

$f_n = f_{n-1} + f_{n-2}$ call fibonacci(4)

procedure *fibonacci* (*4*: nonnegative integer)
if *n = 0* **then return** *0*
else if *n =1* **then return** *1*
else
    $f_{prev} := 1,\ f_{pprev} := 0$
    for i:=2 **to** n
        $f := f_{prev} + f_{pprev},\ f_{prev} := f,\ f_{pprev} := f_{prev}$
return *f*
{output: nth Fibonacci number}

f_{pprev} : 1

f_{prev} : 1

f : 1

i = 2

5.4 Recursive algorithms

Fibonacci numbers

0 1 1 2 3 5 8 13 21 44 65 ...

f_0 f_1 f_2 f_3 f_4 f_5 …

$f_n = f_{n-1} + f_{n-2}$ call fibonacci(4)

procedure *fibonacci* (*4*: nonnegative integer)
if *n = 0* **then return** *0*
else if *n =1* **then return** *1*
else
    $f_{prev} := 1,\ f_{pprev} := 0$
    for i:=2 **to** n
        $f := f_{prev} + f_{pprev},\ f_{prev} := f,\ f_{pprev} := f_{prev}$
return *f*
{output: nth Fibonacci number}

f_{pprev} : 1

f_{prev} : 1

f : 1

i = 3

5.4 Recursive algorithms

Fibonacci numbers

0 1 1 2 3 5 8 13 21 44 65 ...

f_0 f_1 f_2 f_3 f_4 f_5 …

$f_n = f_{n-1} + f_{n-2}$ call fibonacci(4)

procedure *fibonacci* (*4*: nonnegative integer)
if *n = 0* **then return** *0*
else if *n =1* **then return** *1*
else
    $f_{prev} := 1,\ f_{pprev} := 0$
    for i:=2 **to** n
        $f := f_{prev} + f_{pprev},\ f_{prev} := f,\ f_{pprev} := f_{prev}$
return *f*
{output: nth Fibonacci number}

f_{pprev} : 1

f_{prev} : 1

f : 1+1=2

i = 3

5.4 Recursive algorithms

Fibonacci numbers

0 1 1 2 3 5 8 13 21 44 65 ...

f_0 f_1 f_2 f_3 f_4 f_5 …

$f_n = f_{n-1} + f_{n-2}$ call fibonacci(4)

procedure *fibonacci* (*4*: nonnegative integer)
if *n = 0* **then return** *0*
else if *n =1* **then return** *1*
else
    $f_{prev} := 1,\ f_{pprev} := 0$
    for i:=2 **to** n
        $f := f_{prev} + f_{pprev},\ f_{prev} := f,\ f_{pprev} := f_{prev}$
return *f*
{output: n^{th} Fibonacci number}

f_{pprev} : 1

f_{prev} : 2

f : 2

i = 3

5.4 Recursive algorithms

Fibonacci numbers

0 1 1 2 3 5 8 13 21 44 65 ...

f_0 f_1 f_2 f_3 f_4 f_5 …

$f_n = f_{n-1} + f_{n-2}$ call fibonacci(4)

procedure *fibonacci* (*4*: nonnegative integer)
if *n = 0* **then return** *0*
else if *n =1* **then return** *1*
else
    $f_{prev} := 1,\ f_{pprev} := 0$
    for i:=2 **to** n
        $f := f_{prev} + f_{pprev},\ f_{prev} := f,\ f_{pprev} := f_{prev}$
return *f*
{output: nth Fibonacci number}

f_{pprev} : 1

f_{prev} : 2

f : 2

i = 4

5.4 Recursive algorithms

Fibonacci numbers

0 1 1 2 3 5 8 13 21 44 65 ...

f_0 f_1 f_2 f_3 f_4 f_5 …

$f_n = f_{n-1} + f_{n-2}$ call fibonacci(4)

procedure *fibonacci* (*4*: nonnegative integer)
if *n = 0* **then return** *0*
else if *n =1* **then return** *1*
else
    $f_{prev} := 1,\ f_{pprev} := 0$
    for i:=2 **to** n
        $f := f_{prev} + f_{pprev},\ f_{prev} := f,\ f_{pprev} := f_{prev}$
return *f*
{output: nth Fibonacci number}

f_{pprev} : 1

f_{prev} : 2

f : 2+1=3

i = 4

5.4 Recursive algorithms

Fibonacci numbers

0 1 1 2 3 5 8 13 21 44 65 ...

f_0 f_1 f_2 f_3 f_4 f_5 …

$f_n = f_{n-1} + f_{n-2}$ call fibonacci(4)

procedure *fibonacci* (*4*: nonnegative integer)
if *n = 0* **then return** *0*
else if *n =1* **then return** *1*
else
    $f_{prev} := 1,\ f_{pprev} := 0$
    for i:=2 **to** n
        $f := f_{prev} + f_{pprev},\ f_{prev} := f,\ f_{pprev} := f_{prev}$
return *f*
{output: nth Fibonacci number}

f_{pprev} : 2

f_{prev} : 3

f : 3

i = 4

5.4 Recursive algorithms

Fibonacci numbers

0 1 1 2 3 5 8 13 21 44 65 ...

f_0 f_1 f_2 f_3 f_4 f_5 …

$f_n = f_{n-1} + f_{n-2}$ call fibonacci(4)

procedure *fibonacci* (*4*: nonnegative integer)
if *n = 0* **then return** *0*
else if *n =1* **then return** *1*
else
    $f_{prev} := 1,\ f_{pprev} := 0$
    for i:=2 **to** n
        $f := f_{prev} + f_{pprev},\ f_{prev} := f,\ f_{pprev} := f_{prev}$
return *f*
{output: nth Fibonacci number}

f_{pprev} : 2

f_{prev} : 3

f : 3

i = 4

5.4 Recursive algorithms

Fibonacci numbers

0 1 1 2 3 5 8 13 21 44 65 ...
f_0 f_1 f_2 f_3 f_4 f_5 ...

procedure *fibonacci_rec* (*n*: nonnegative integer)
if *n = 0* **then return** *0*
else if *n =1* **then return** *1*
else return *fibonacci_rec(n-1)* + *fibonacci_rec(n-2)*
{output: n^{th} Fibonacci number}

5.4 Recursive algorithms: Fibonacci numbers

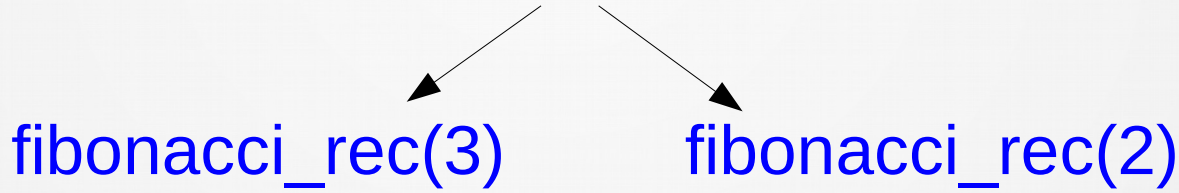
procedure *fibonacci_rec* (*n*: nonnegative integer)
if *n = 0* **then return** *0*
else if *n =1* **then return** *1*
else return *fibonacci_rec(n-1) + fibonacci_rec(n-2)*
{output: n^{th} Fibonacci number}

call fibonacci_rec(4):

5.4 Recursive algorithms: Fibonacci numbers

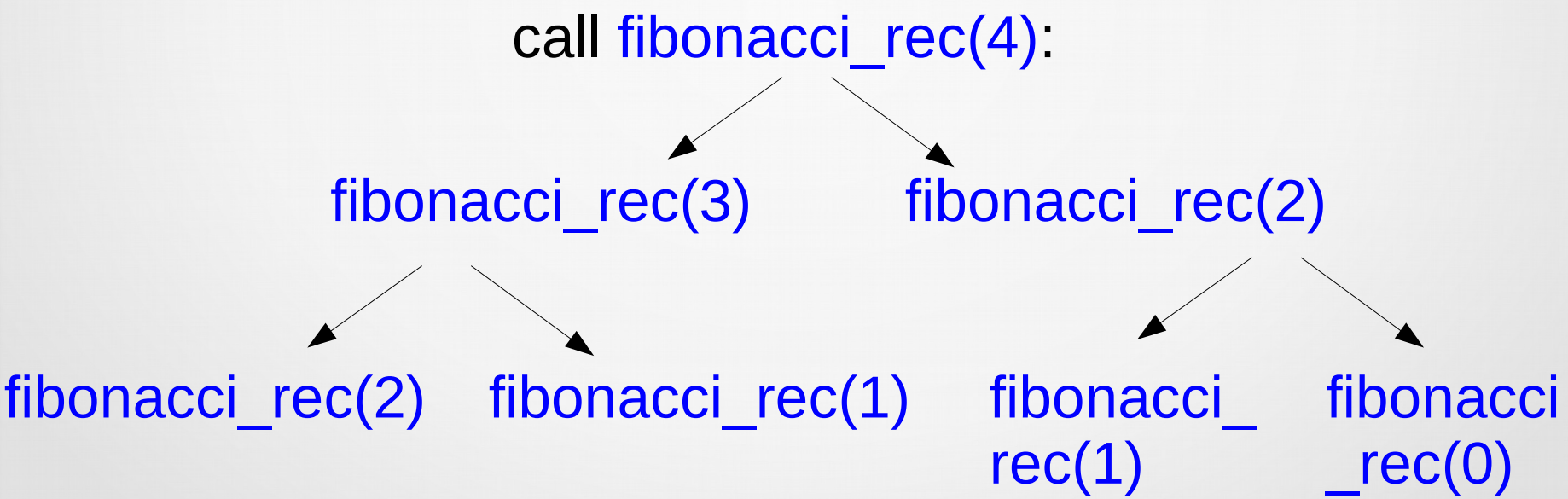
procedure *fibonacci_rec* (*n*: nonnegative integer)
if *n = 0* **then return** *0*
else if *n =1* **then return** *1*
else return *fibonacci_rec(n-1) + fibonacci_rec(n-2)*
{output: n^{th} Fibonacci number}

call fibonacci_rec(4):



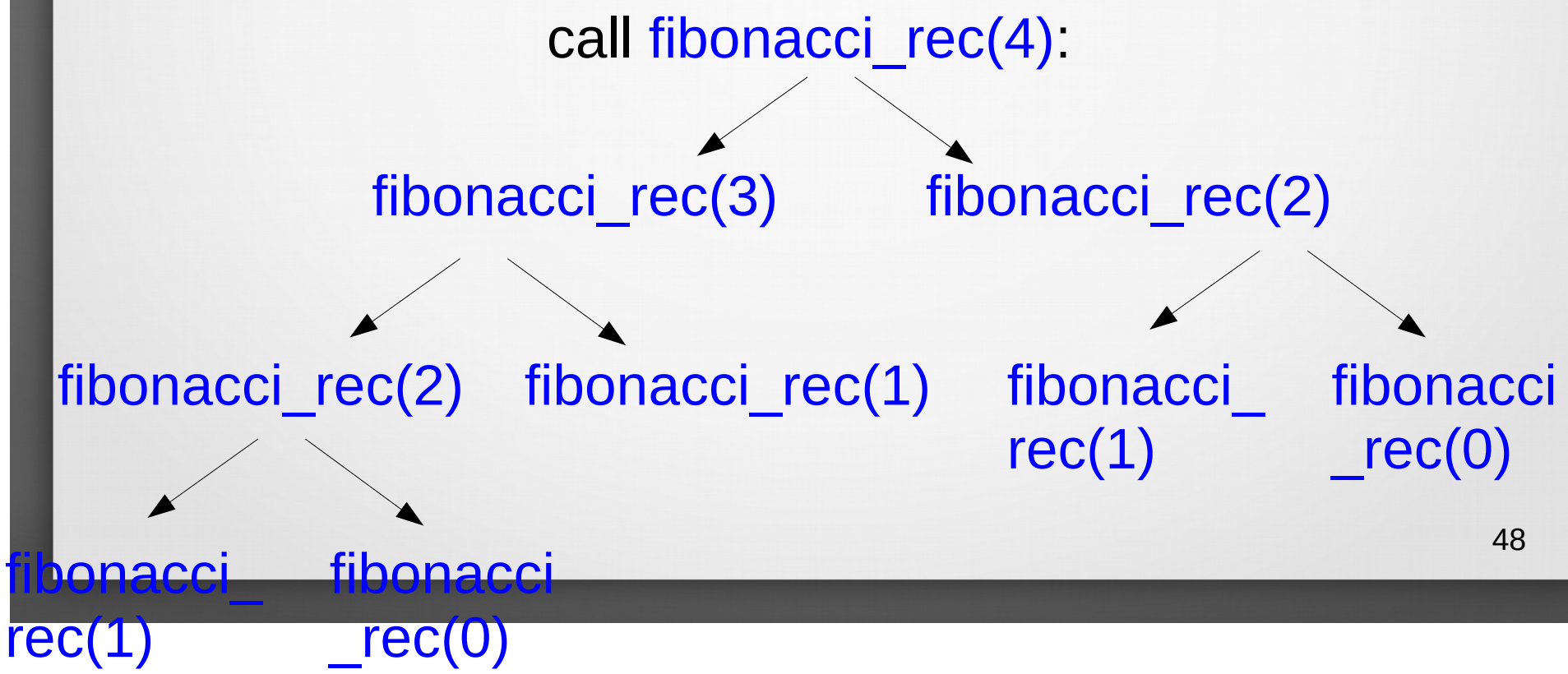
5.4 Recursive algorithms: Fibonacci numbers

procedure *fibonacci_rec* (*n*: nonnegative integer)
if *n = 0* **then return** *0*
else if *n =1* **then return** *1*
else return *fibonacci_rec(n-1) + fibonacci_rec(n-2)*
{output: n^{th} Fibonacci number}



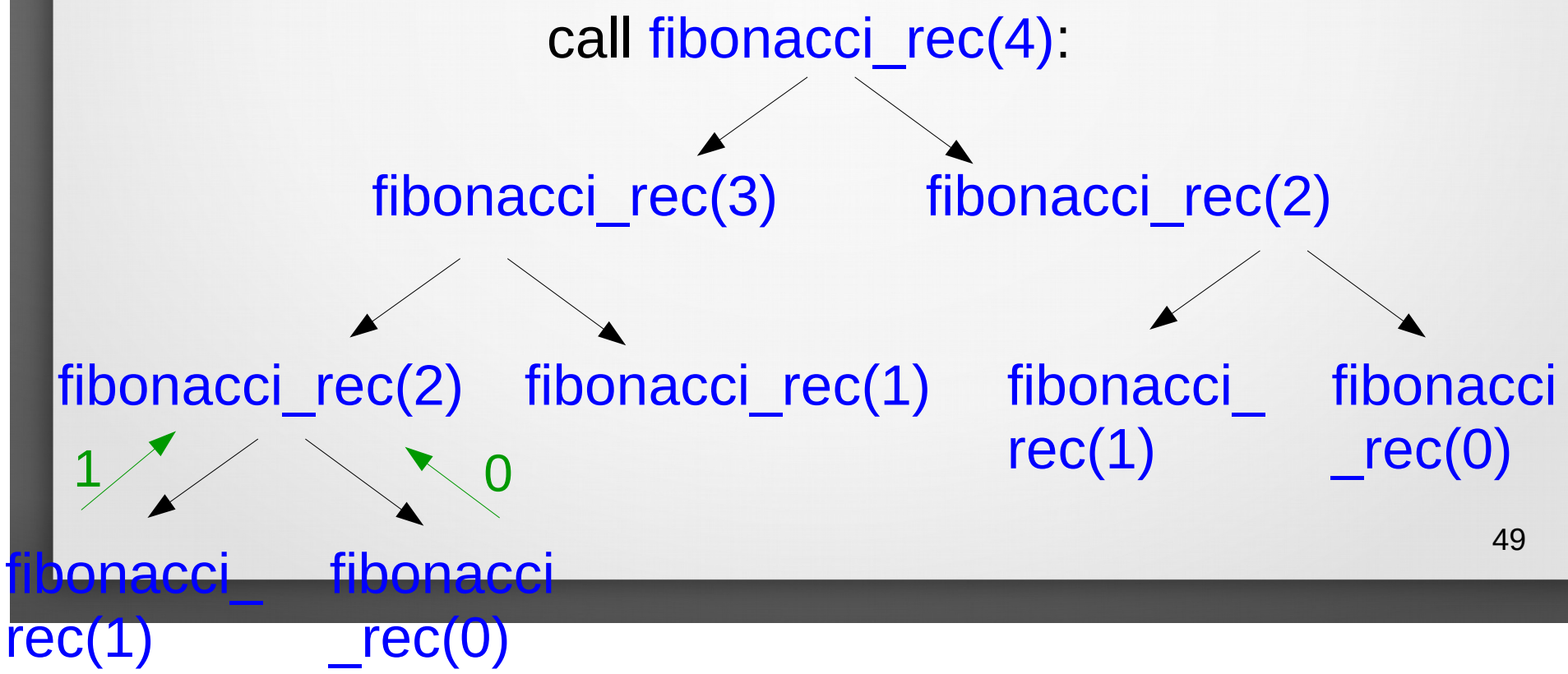
5.4 Recursive algorithms: Fibonacci numbers

procedure *fibonacci_rec* (*n*: nonnegative integer)
if *n = 0* **then return** *0*
else if *n =1* **then return** *1*
else return *fibonacci_rec(n-1) + fibonacci_rec(n-2)*
{output: n^{th} Fibonacci number}



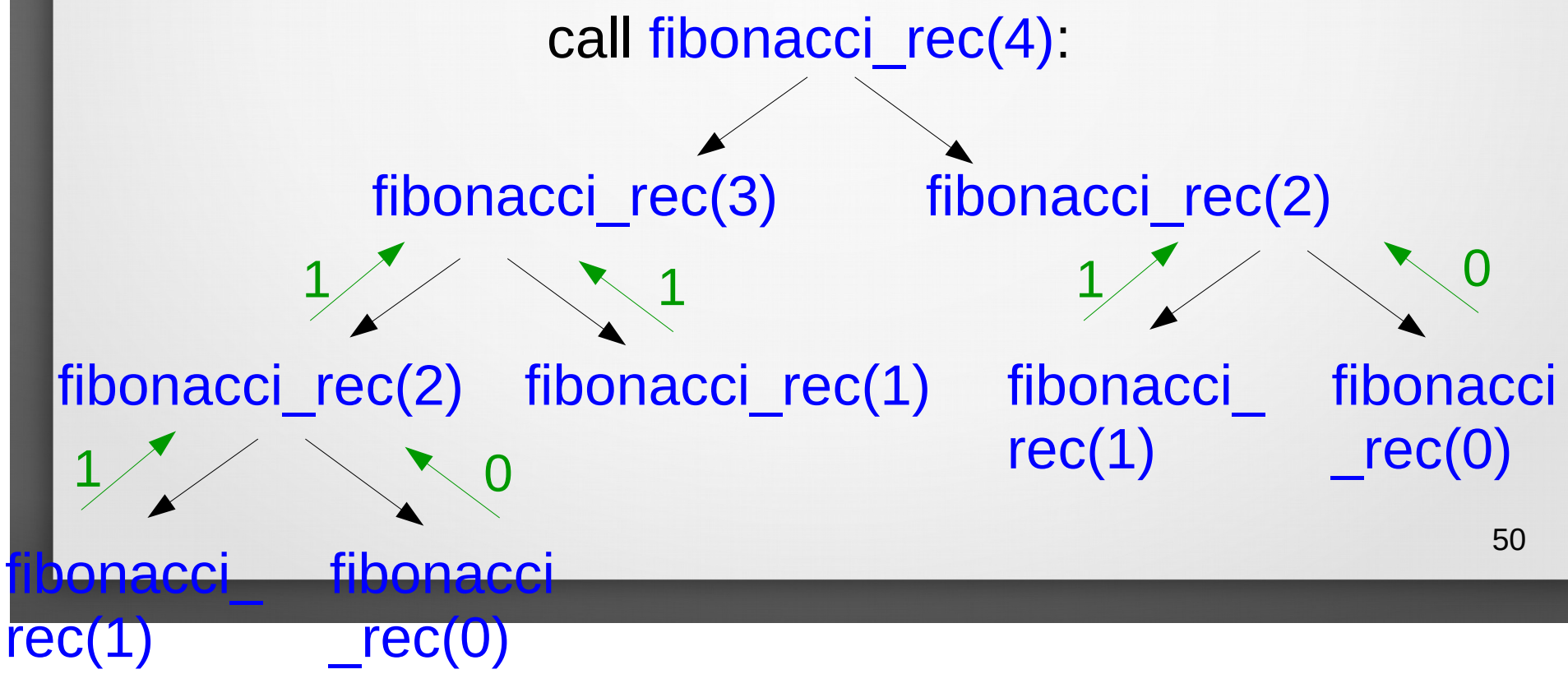
5.4 Recursive algorithms: Fibonacci numbers

procedure *fibonacci_rec* (*n*: nonnegative integer)
if *n = 0* **then return** *0*
else if *n =1* **then return** *1*
else return *fibonacci_rec(n-1) + fibonacci_rec(n-2)*
{output: n^{th} Fibonacci number}



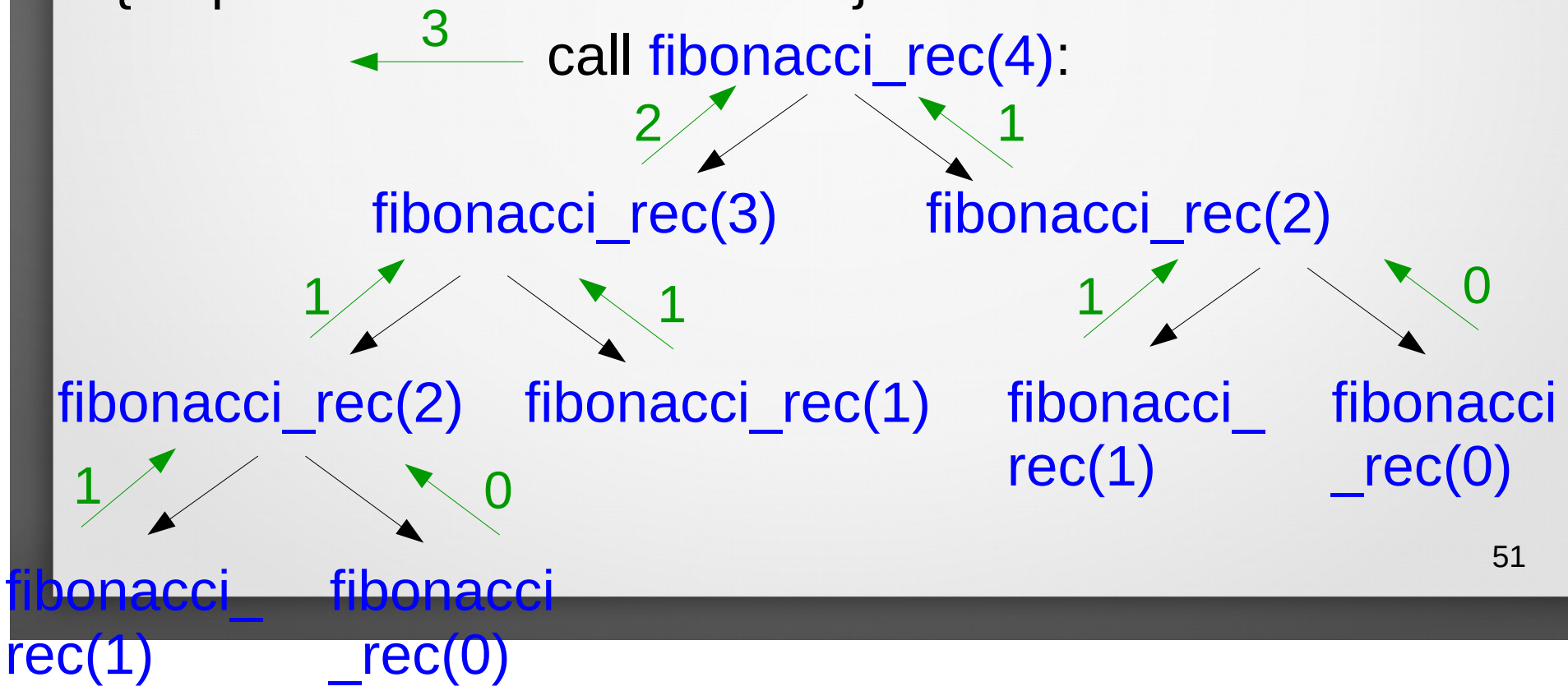
5.4 Recursive algorithms: Fibonacci numbers

procedure *fibonacci_rec* (*n*: nonnegative integer)
if *n = 0* **then return** *0*
else if *n =1* **then return** *1*
else return *fibonacci_rec(n-1) + fibonacci_rec(n-2)*
{output: n^{th} Fibonacci number}



5.4 Recursive algorithms: Fibonacci numbers

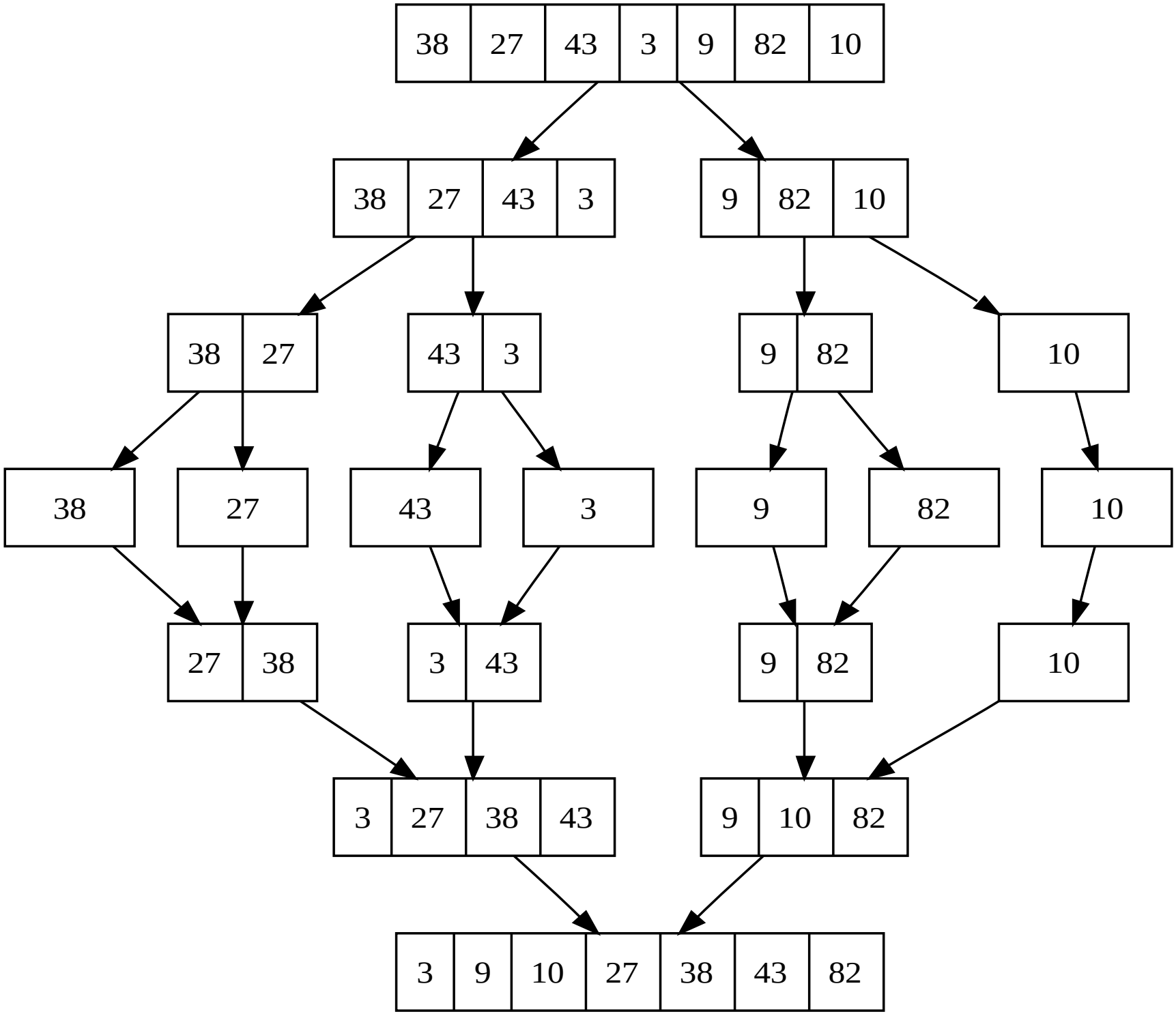
procedure *fibonacci_rec* (*n*: nonnegative integer)
if *n = 0* **then return** *0*
else if *n =1* **then return** *1*
else return *fibonacci_rec(n-1) + fibonacci_rec(n-2)*
{output: n^{th} Fibonacci number}



5.4 Recursive algorithms: Merge sort

procedure *mergesort_rec* ($L=a_1,a_2,...,a_n$: integers)
if $n > 1$ **then**
    $m := \lfloor n/2 \rfloor$
    $L_1 := a_1, a_2, ..., a_m$, $L_2 := a_{m+1},...,a_n$
    $L := merge(mergesort(L_1),mergesort(L_2))$
{output: *L* is sorted in non-decreasing order}

procedure *merge*(L_1,L_2: sorted lists)
L := empty list
while L_1 and L_2 are both non-empty
    remove the smaller of first elements of L_1 and L_2
    put it at the right end of list *L*
    if a list becomes empty, then remove all the elements from the other list and append them to *L*
return *L*


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

5.4 Recursive algorithms: Powerset

procedure *powerSet*(*A*: a set)
if $A = \varnothing$ **return** $\{\varnothing\}$
else
    select an element $a \in A$
    $A_1 := A - \{a\}$
    $P := powerSet(A_1)$ // the recursive call
    for each $S \in P$
        add $S \cup \{a\}$ to P

    return *P*
{output: The power set of *A*.}

5.4 Recursive algorithms: Powerset

procedure *powerSet*(*A*: a set)
if $A = \varnothing$ **return** $\{\varnothing\}$
else
    select an element $a \in A$
    $A_1 := A - \{a\}$
    $P := powerSet(A_1)$
    for each $S \in P$
        add $S \cup \{a\}$ to P

    return *P*
{output: The power set of *A*.}

5.4 Recursive algorithms: Powerset

procedure *powerSet*(*A*: a set)
if $A = \varnothing$ **return** $\{\varnothing\}$
else
    select an element $a \in A$
    $A_1 := A - \{a\}$
    $P := powerSet(A_1)$
    for each $S \in P$
        add $S \cup \{a\}$ to P

    return *P*
{output: The power set of *A*.}

powerSet({1,2}) call:

5.4 Recursive algorithms: Powerset

procedure *powerSet*(*A*: a set)
if $A = \varnothing$ **return** $\{\varnothing\}$
else
    select an element $a \in A$
    $A_1 := A - \{a\}$
    $P := powerSet(A_1)$
    for each $S \in P$
        add $S \cup \{a\}$ to P

    return *P*
{output: The power set of *A*.}

powerSet({1,2}) call:

$a = 1$
$A_1 = \{2\}$
$P = powerSet(\{2\})$

5.4 Recursive algorithms: Powerset

procedure *powerSet*(*A*: a set)
if $A = \varnothing$ **return** $\{\varnothing\}$
else
    select an element $a \in A$
    $A_1 := A - \{a\}$
    $P := powerSet(A_1)$
    for each $S \in P$
        add $S \cup \{a\}$ to P

    return *P*
{output: The power set of *A*.}

powerSet({1,2}) call:

$a = 1$
$A_1 = \{2\}$
$P = powerSet(\{2\})$

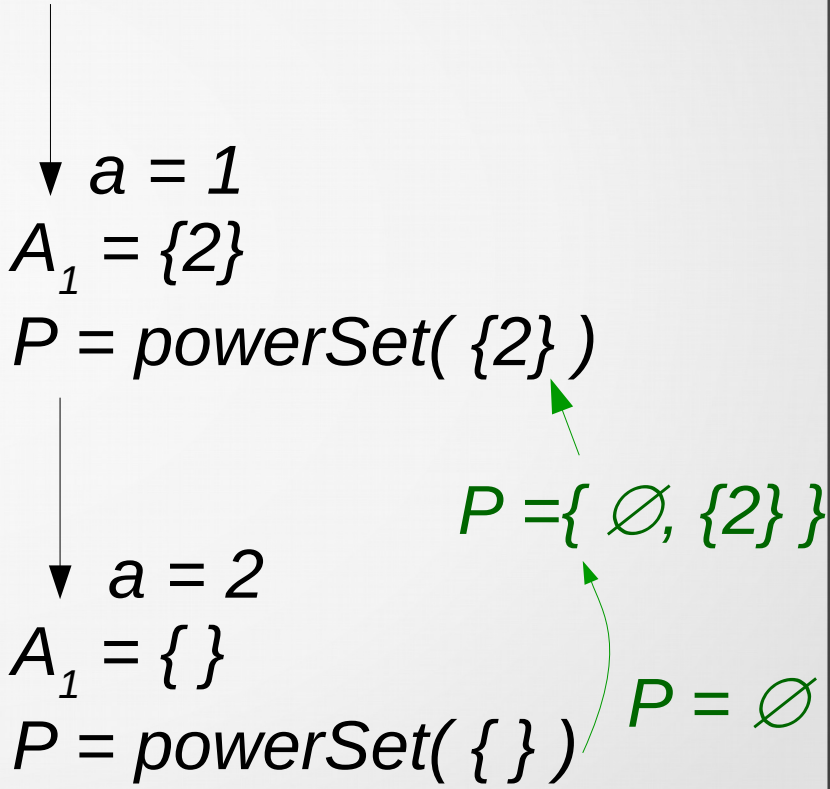
$a = 2$
$A_1 = \{ \}$
$P = powerSet(\{ \})$

5.4 Recursive algorithms: Powerset

procedure *powerSet*(*A*: a set)
if $A = \varnothing$ **return** $\{\varnothing\}$
else
    select an element $a \in A$
    $A_1 := A - \{a\}$
    $P := powerSet(A_1)$
    for each $S \in P$
        add $S \cup \{a\}$ to P

    return *P*
{output: The power set of *A*.}

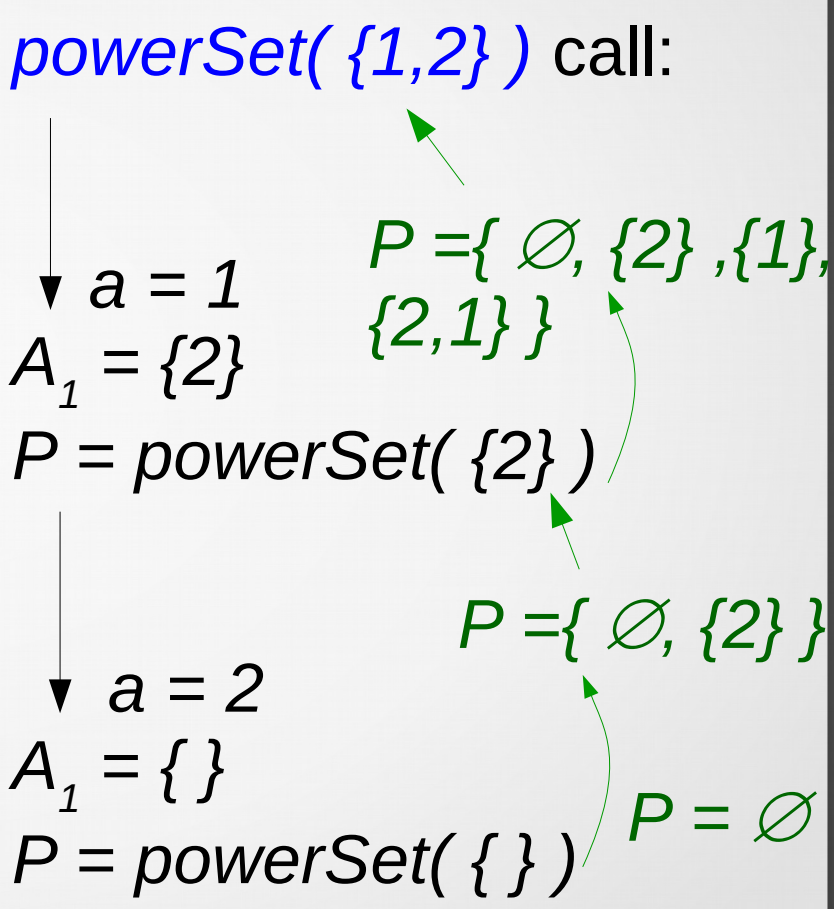
powerSet({1,2}) call:



5.4 Recursive algorithms: Powerset

procedure *powerSet*(*A*: a set)
if $A = \varnothing$ **return** $\{\varnothing\}$
else
    select an element $a \in A$
    $A_1 := A - \{a\}$
    $P := powerSet(A_1)$
    for each $S \in P$
        add $S \cup \{a\}$ to P

    return *P*
{output: The power set of *A*.}



5.4 Recursive algorithms: Powerset

procedure *powerSet*(*A*: a set)
if $A = \varnothing$ **return** $\{\varnothing\}$
else
    select an element $a \in A$
    $A_1 := A - \{a\}$
    $P := powerSet(A_1)$
    for each $S \in P$
        add $S \cup \{a\}$ to P

    return *P*
{output: The power set of *A*.}

$P = \{ \varnothing, \{2\}, \{1\}, \{2,1\} \}$

powerSet({1,2}) call:

$a = 1$
$A_1 = \{2\}$
$P = powerSet(\{2\})$

$a = 2$
$A_1 = \{ \}$
$P = powerSet(\{ \})$

$P = \varnothing$

$P = \{ \varnothing, \{2\} \}$

$P = \{ \varnothing, \{2\}, \{1\}, \{2,1\} \}$