



University of  
Zurich <sup>UZH</sup>

S3IT

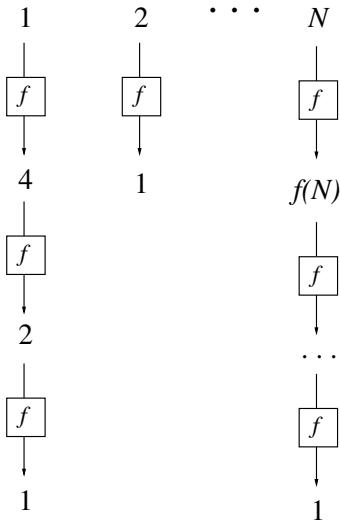
# Dynamic and Unbounded Sequences of Tasks

Riccardo Murri <[riccardo.murri@uzh.ch](mailto:riccardo.murri@uzh.ch)>

*S3IT: Services and Support for Science IT*

University of Zurich

# The $3n+1$ conjecture, a fictitious use case



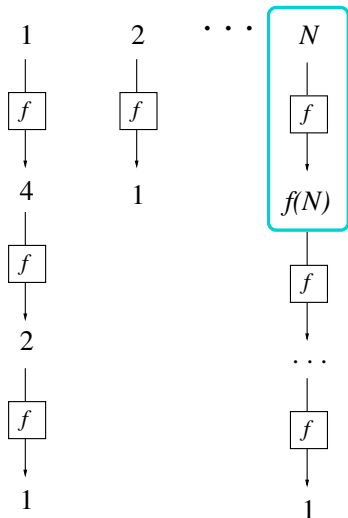
Define a function  $f$ , for  $n$  positive integer:

- ▶ if  $n$  is even, then  $f(n) = n/2$ ,
- ▶ if  $n$  is odd, then  $f(n) = 3n + 1$ ,

For every positive integer  $n$ , form the sequence  $S(n)$ :  $n \rightarrow f(n) \rightarrow f(f(n)) \rightarrow f(f(f(n))) \rightarrow \dots$

**Conjecture:** For every positive integer  $n$ , the sequence  $S(n)$  eventually hits 1.

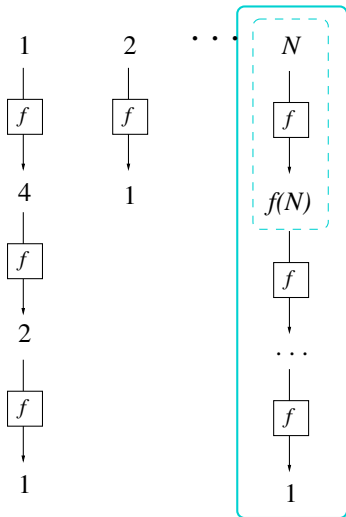
# The $3n+1$ conjecture, I



A computational job  $F(n, k)$ , applies function  $f$  to the result of  $F(n, k - 1)$ .

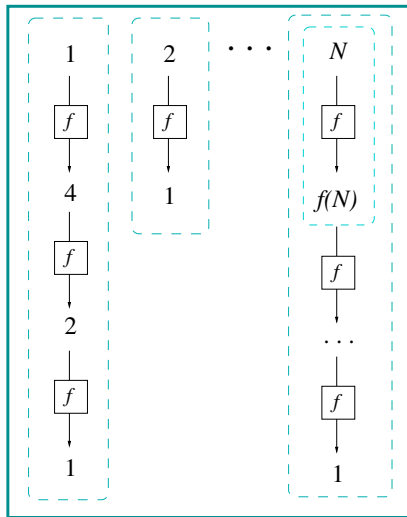
(With  $F(n, 0) = n$ .)

## The $3n+1$ conjecture, II



A sequence  $H(n)$  of jobs computes the chain  $n \rightarrow f(n) \rightarrow \dots \rightarrow 1$ .

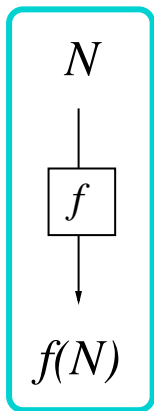
## The $3n+1$ conjecture, III



Run one sequence  $H(n)$   
per each  $n = 1, \dots, N$ .

They all can run in  
**parallel.**

## The $3n+1$ conjecture, IV

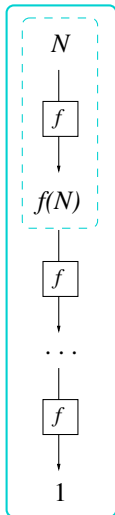


Let's define the simple application that computes  $f$ :

```
class HotpoApplication(Application):  
    def __init__(self, n):  
        Application.__init__(  
            self,  
            arguments = (['/usr/bin/expr'] +  
                # run 'expr n / 2' if n even  
                [n, '/', n] if n % 2 == 0  
                # 'expr 1 + 3 * n' if n odd  
                else [1, '+', 3, '*', n]),  
            stdout = "stdout.txt",  
            # ...  
        )
```

## The $3n+1$ conjecture, V

Now string together applications to compute a single sequence:



```
from gc3libs.workflow \
    import SequentialTaskCollection as Seq

class HotpoSequence(Seq):

    def __init__(self, n):
        # compute first iteration of f
        SequentialTask.__init__(self,
            [ HotpoApplication(n) ])

    def next(self, k):
        last = self.tasks[k].result
        if last == 1:
            return TERMINATED
        else:
            self.tasks.append(HotpoApplication(last))
            return RUNNING
```

## The next () method in SequentialTaskCollection

The **next ()** method is called whenever a task in the sequence has turned to **TERMINATED** state.

```
from gc3libs.workflow \
    import SequentialTaskCollection as

class HotpoSequence(Seq):

    # ...

    def next(self, k):
        last = self.tasks[k].result
        if last == 1:
            return TERMINATED
        else:
            self.tasks.append(
                HotpoApplication(last))
            return RUNNING
```



## The next () method in SequentialTaskCollection

The second argument to `next ()` is the index (within `self.tasks`) of the task that just finished.

```
from gc3libs.workflow \
    import SequentialTaskCollection as

class HotpoSequence(Seq):

    # ...

    def next(self, k):
        last = self.tasks[k].result
        if last == 1:
            return TERMINATED
        else:
            self.tasks.append(
                HotpoApplication(last))
            return RUNNING
```

## The next () method in SequentialTaskCollection

You can access all  
attributes of tasks  
that are already done.

```
from gc3libs.workflow \
    import SequentialTaskCollection as

class HotpoSequence(Seq):

    # ...

    def next(self, k):
        last = self.tasks[k].result
        if last == 1:
            return TERMINATED
        else:
            self.tasks.append(
                HotpoApplication(last))
            return RUNNING
```

## The next () method in SequentialTaskCollection

Returning the state  
TERMINATED  
interrupts the  
sequence: no other  
tasks from this  
collection will be run.

```
from gc3libs.workflow \
    import SequentialTaskCollection as

class HotpoSequence(Seq):

    # ...

    def next(self, k):
        last = self.tasks[k].result
        if last == 1:
            return 'TERMINATED'
        else:
            self.tasks.append(
                HotpoApplication(last))
            return 'RUNNING'
```

## The next () method in SequentialTaskCollection

```
from gc3libs.workflow \
    import SequentialTaskCollection as

class HotpoSequence(Seq):
```

It is entirely possible  
to modify the  
SequentialTaskCollection  
and add (or remove)  
tasks.

```
# ...

def next(self, k):
    last = self.tasks[k].result
    if last == 1:
        return 'TERMINATED'
    else:
        self.tasks.append(
            HotpoApplication(last))
    return 'RUNNING'
```

## The next () method in SequentialTaskCollection

Returning state  
RUNNING makes the  
sequence continue  
with task  $k+1$

```
from gc3libs.workflow \
    import SequentialTaskCollection as

class HotpoSequence(Seq):

    # ...

    def next(self, k):
        last = self.tasks[k].result
        if last == 1:
            return 'TERMINATED'
        else:
            self.tasks.append(
                HotpoApplication(last))
            return 'RUNNING'
```

## The next () method in SequentialTaskCollection

Alternatively, you can **return** a number  $j$  less than  $k$ , meaning that the sequence will rewind to the  $j$ -th task and continue running from there.

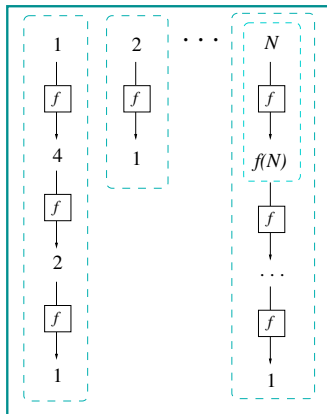
```
from gc3libs.workflow \
    import SequentialTaskCollection as

class HotpoSequence(Seq):

    # ...

    def next(self, k):
        last = self.tasks[k].result
        if last == 1:
            return 'TERMINATED'
        else:
            self.tasks.append(
                HotpoApplication(last))
            return 'RUNNING'
```

## The $3n+1$ conjecture, VI



Parallel tasks are independent by definition, so it's even easier to create a collection:

```
tasks = ParallelTaskCollection([
    HotpoSequence(n)
    for n in range(1, N)
])
```

We can run such a collection like any other Task.

## Exercise 11.A:

Fill in the missing parts and write a `hotpo.py` session-based script that:

- ▶ takes a single integer parameter  $N$  on the command-line:

```
$ python hotpo.py 42
```

- ▶ computes all the “ $3n + 1$ ” sequences of numbers 1 up to  $N$  in parallel,
- ▶ prints a final statement that the Collatz conjecture is verified up to  $N$  (or —who knows— not?)