Algorithms and their composition in GenLab

Samuel Thiriot

June 15, 2014

Contents

1 Basics of Genlab and simple example **5**
- 1.1 Basic concepts in GenLab . . . 5
  - 1.1.1 Workflow . . . 5
  - 1.1.2 Algorithm . . . 5
  - 1.1.3 Links between algorithms . . . 5
  - 1.1.4 Parameters . . . 5
- 1.2 Execution of workflows in GenLab . . . 5
  - 1.2.1 What happens at execution time ? . . . 5
- 1.3 Tutorial: plot random data . . . 6
  - 1.3.1 create the workflow . . . 6
  - 1.3.2 populate the workflow . . . 6
  - 1.3.3 run the workflow . . . 6
  - 1.3.4 tune parameters of algorithms . . . 6
- 1.4 Tutorial: use the same output several times . . . 6
- 1.5 Tutorial: merge several outputs as one input . . . 6

2 Advanced concepts: container algorithms **7**
- 2.1 Motivation and examples . . . 7
  - 2.1.1 “foor” loop example . . . 7
- 2.2 Inputs out of the container connected inside the container . . . 7
  - 2.2.1 Rationale . . . 7
  - 2.2.2 What happens . . . 7
- 2.3 All the possible and impossible combinations . . . 8
  - 2.3.1 A linked to C inside B: OK, principle of multiple reuse . . . 8
  - 2.3.2 A linked to D inside C inside B: should be OK, principle of multiple reuse . . . 8
  - 2.3.3 C inside A linked to D inside B: not possible . . . 8
  - 2.3.4 B inside A linked to C: OK, principle of reduction . . . 9

3 Reference **11**
- 3.1 Algorithms executions . . . 11
  - 3.1.1 Atomic algorithm . . . 11
  - 3.1.2 Constant algorithm (or algorithm without inputs) . . . 11
  - 3.1.3 Container algorithm . . . 11
  - 3.1.4 Reduce algorithm . . . 12

4 TODO reuse **13**
4.1 Algorithms 13
4.1.1 Algorithm: definition 13
4.1.2 Types of algorithms 13
4.2 Input and outputs 14
4.2.1 oneshot vs. continuous 14
4.2.2 multiple inputs vs. unique inputs 14
4.3 Examples of workflows 15
4.3.1 simple workflow: generate random data and plot it 15
4.3.2 use case: debugging workflows 15
4.3.3 simple workflow: generate a network and display it 15
4.4 execution options 15

Chapter 1

Basics of Genlab and simple example

1.1 Basic concepts in GenLab

1.1.1 Workflow

In genlab, users create workflows. A workflow may be visualized and edited as a graph of boxes - the algorithms - linked together by arrows. Each box is an algorithm which takes inputs, exports outputs, and maybe have associated parameters. A workflow is made to be executed; each box will be ran after each other (or in parallel to each other), the output of each box being used as an input of the boxes it is connected to.

Execution of a workflow means GenLab will execute the graph of computations depending on their links (the structure of the workflow).

1.1.2 Algorithm

1.1.3 Links between algorithms

1.1.4 Parameters

1.2 Execution of workflows in GenLab

1.2.1 What happens at execution time ?

To execute a workflow:

- the workflow is checked
- the workflow is automatically converted to an execution graph
- the execution graph is actually executed (either locally or remotly)

1.3 Tutorial: plot random data

1.3.1 create the workflow

Create a novel project. Create a novel workflow.

1.3.2 populate the workflow

Add a table generator. Add a plot. Link them.

1.3.3 run the workflow

At runtime, each algorithm will run. You can see several windows opened. One is the messages log for this execution. It will always open at runtime. It contains messages about what happened during the computation of the workflow. Each algorithm may report notices, warnings or errors to help you to understand what happened during the workflow execution.

Tune parameters of dispkay

Note that it's not possible to view the intermediate or final results afterwards: the data is cleaned during computation.

1.3.4 tune parameters of algorithms

run again

1.4 Tutorial: use the same output several times

One may add a file save of this table.

1.5 Tutorial: merge several outputs as one input

Most of inputs of algorithms accept only one connection.

But in some cases it may be usefull to merge several ouputs as one unique input. Specific algorithms accept several arrows for an input.

Example of fusionning several inputs to display them

Example of funsionning several graphs to display them.

Chapter 2

Advanced concepts: container algorithms

2.1 Motivation and examples

use case: be willing to run one thousands generations of graphs, analyse their statistical properties, and compare them with some statistical processes (like: mean and standard deviation).

2.1.1 "foor" loop example

Add a for loop in the workflow. Inside it, create the algorithms for this task. Run it. Observe that:

- The algorithms inside the containers are going to be executed several times.
- The algorithms inside the containers are sometimes executing in parallel.
-

2.2 Inputs out of the container connected inside the container

2.2.1 Rationale

We executed the workflow depicted in figure TODO. In this example, each element of this workflow was executed several times. But some are useless. We may have placed these algorithms out of the loop.

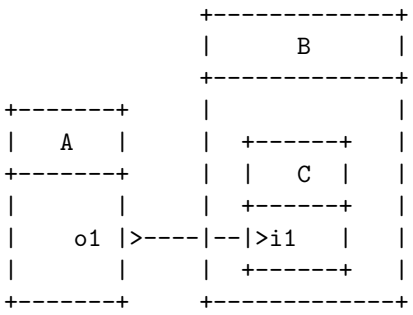
2.2.2 What happens

When preparing the execution of the workflow, GenLab will detect the presence of these links which are connected from an output

2.3 All the possible and impossible combinations

2.3.1 A linked to C inside B: OK, principle of multiple reuse

Principle



Behaviour

What happens: B has for input (A, a1). Will execute A *one time*. *One* result of a1 transmitted to B. Then B will run. B will execute C several times. For each run, it will transmit *the same value* a1 as an input of C.

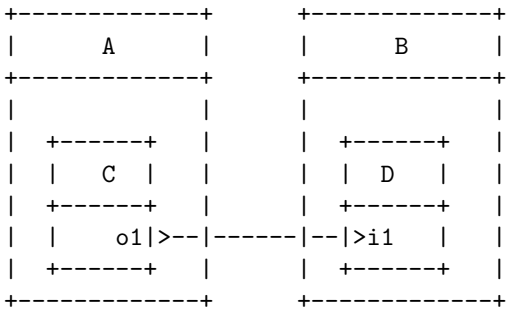
Example

Example: a random generator. It will run once. All subsequent iterations will lead to differnt results.

2.3.2 A linked to D inside C inside B: should be OK, principle of multiple reuse

TODO !!!

2.3.3 C inside A linked to D inside B: not possible



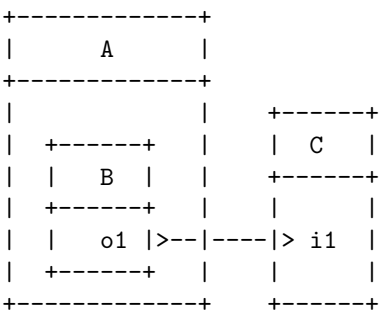
This behaviour can not be analyzed by GenLab:

- A executes C several times
- C export o1, o1, o1 and o1

- when should B start ? Which version of o1 should be transmitted to (D, i1) ?

2.3.4 B inside A linked to C: OK, principle of reduction

Principle



An algorithm is said reduce if it is able to take several successive values from an arrow in an input.

When an algorithm is ran several successive times, we need a way to *reduce* these various computations to one.

Behaviour

What happens:

- A is ready, B is ready, C is waiting for inputs
- A runs. It creates many parallel executions of B
- each execution of B is ran. each outputs a value for (B, o1) (C, i1) receives all these inputs, and reduces them.

The actual reduction behaviour depends on the algorithm.

Example: "append table" export

The "append table" algoritm receives numeric values as inputs, and appends them into a table. When connected to a container, it will create on row per run. For each input of one run, it will fill a corresponding cell.

Other reduce algorithms are more simple. The "console display" algorithm just displays everything incoming, wathever its origin.

Chapter 3

Reference

3.1 Algorithms executions

3.1.1 Atomic algorithm

Preparation

Execution

- created in state "pending"
- everytime an input becomes available: check if all inputs are OK; if all inputs available, shift to state ready
- will be executed at some point
- once executed, reads all the inputs, processes them, emits outputs, shift to state terminated

3.1.2 Constant algorithm (or algorithm without inputs)

Preparation: none (always ok)

Execution:

- created in state "available"
- will be executed at some point
- once executed, emits outputs, shift to state terminated

3.1.3 Container algorithm

Preparation:

Execution

- Detect all the incoming links coming from an external element. Listen to them.
- Detect all the outgoing links going to outside.
- TODO

3.1.4 Reduce algorithm

Preparation:

Execution

- at creation, prepare to receive incoming data even before startup of this task
- at each execution start, prepare to receive this input
-

Chapter 4

TODO reuse

4.1 Algorithms

4.1.1 Algorithm: definition

In genlab, an *algorithm is a process which can be instanciated inside workflows; when ran, an algorithm will, for a given set of inputs and based on parameters, compute outputs.* Algorithms have several properties:

- no side effect (TODO effet de bord): during and after the computation, the algorithm exports outputs and messages, but did not changes anything else in the environment (except for filesystem, database or external storage aspects).
-

An algorithm exists at the "meta" level (in the library of modules). Then a module can be instanciated - meaning it was added inside a workflow. Then the workflow, and so the corresponding algorithm instances, can be executed.

4.1.2 Types of algorithms

TODO Deterministic/stochastic

TODO graphic / nongraphic

Atomic vs. Container algorithms

An algorithm is either atomic or container.

An atomic algorithm is the simplest version of an algorithm.

A *container algorithm* may contain instances of algorithms, exactly as workflows may contain algorithm instances.

Constant algorithms

Constant algorithms have no input, no parameter, and always produce the very same output. Examples of constants include integer values or boolean values.

4.2 Input and outputs

4.2.1 oneshot vs. continuous

The most straightforward input and output is one-shot, meaning there will be one unique activity on these input/outputs during an execution.

- During an execution, a one-shot input will receive only one value sent by another algorithm.
- During an execution, a one-shot ouput will produce only one value at the end of the execution of the algorithm.

One-shot outputs may not be enough for certain types of algorithms:

- when an algorithm executes during very long times, the user may be willing to observe intermediate results, or to retrieve intermediate results in case of failure of the algorithm
- when an algorithm processes large amounts of data, one does not wants the rest of the chain to wait for the whole output
-

In these cases:

- a continuous output is an output which may sent results during the computation
- a continuous input is an input which accepts several different inputs

4.2.2 multiple inputs vs. unique inputs

Most inputs accept only one input. For instance, to generate a table, you may generate a table with either 12 or 23, but this would be a non sense to transmit both 12 and 23. But some algorithms may accept several different inputs and combine them because this is meaningfull for their task. Typically a console display will take anything as input and print it as soon as possible in output. A graph display may accept several graphs and display them all (TODO ?)

Unique inputs accept only one input arrow. The graphical user interface will not allow you to plug several inputs arrows on it.

This concept should not be confused with continuous vs. one-shot.

4.3 Examples of workflows

4.3.1 simple workflow: generate random data and plot it

(C1, C2) –¿ A –¿ B

When executing the workflow, Genlab will detect how to execute it:

- C1 and C2 will be executed in parallel; maybe C1 will finish first, then C2
- C1 terminates and sends its output to A. A reviews its inputs and analyzes it is still waiting for C2, so it still waits.
- C2 terminates and sends its ouput to A. A reviews its inputs and discovers all the inputs are available. So it switches to state "ready". The runner discovers that and starts the execution of A
- A terminates its computation and sends the result to B.
- B receives a notification of a novel input; B has all inputs ready, so will run.
- B is terminated. No more algorithm is in status running or pending, so the workflow execution is terminated.

4.3.2 use case: debugging workflows

One possibility is to observe possible outcomes. You may always plug a value you are not certain of to a console display, to have it displayed as text in a console.

4.3.3 simple workflow: generate a network and display it

4.4 execution options

A –¿ [B]