Explanation of the Analyzer

Clémence Sebe and George Marchment

université PARIS-SACLAY

General explanation of the analyzer

Main

Calls the method main:
- Retrieve the parameters (for which mode will be used)
- Check at the input address and results directory are given
- Set the current directory to where the data will be saved (directory retrieved with the parameters)
- If SINGLE mode selected (Analyze a single workflow which the address is given by the parameters) :
    - Creating the folder where the data extracted will be saved : the name of which is given to the analyzer by the user with the parameter '*--name*', the default is '*Workflow_Analysis*'.
    - Analyzing the workflow see Workflow
    - Deletes the developer files, if not in dev mode 'T' or 'F' (default 'F')
- If MULTI mode selected (Analyzing a set of workflows in a file : which the address is given by the user)

- IMPORTANT : For multi mode, it only uses the nextflow files found at the root of the folder (for the analysis of DSL1 workflows). So it is important that there aren't any residu DSL2 files in the folder since the analyzer will try to analyze the workflow, the subworkflow and the module files. There can be mains of DSL2 since it is in these files that the DSL2 mode is selected, the analyzer can recognize this. For the analysis of workflows written in DSL2 they have to be given in individual folders (1 workflow => 1 folder), and all the nextflow files in that folder will be analyzed and be considered part of that workflow.
- Declare variables to count the number of workflows which have been analyzed, etc...
- Same but with lists, to save the workflows (the names)
- Retrieve the addresses of the workflows in the folders to analyze (both DSL1 and DSL2)
- For the files found (hence supposed written in DSL1) :
    - Extract the names of the workflows
    - For each workflow
        - Create the new folder to save the data from the analyze
        - Analyzing the workflow see Workflow
        - If there is an Error during the execution of the analyze it is caught and its type is shown
        - Deletes the developer files, if not in dev mode ‘T’ or ‘F’ (default ‘F’)
- For the folders found (hence supposed written in DSL2) :
    - For each workflow
        - Create the new folder to save the data from the analyze
        - Analyzing the workflow see Workflow
        - If there is an Error during the execution of the analyze it is caught and its type is shown
- Save the different statistics if files at the root of the results

Workflow

When initializing a workflow, there are 2 possibilities :

- The address given is a file, we suppose then that it is a workflow written in DSL1 (we still check), these are the steps to extract the structure and the analysis of the processes :
    - Check to see if the workflow is using DSL2 or not
        - Define the pattern used to identify if the workflow is written in DSL2 or not (pattern found in the source code of Nextflow)
        - Create an object of File (see File) type to be able to remove the comments from the code
        - Get the string from the object with comments removed
        - Check if the pattern is used in the string if so the workflow is written in DSL2, otherwise in DSL1. Reminder : a DSL1 workflow is written in one file, DSL2 can be written in multiple files, that is why the user has to be careful of the type of workflows given to the analyzer when using *multi* mode (see here)

        - Analyze the workflow
            - If the workflow is written in DSL1
                - Create an object of type *TypeMainDSL1* and initialize it (full analysis : extract the processes + structure)
            - If the workflow is written in DSL2
                - Raise the error : "*Single file is written in DSL2*". For now the analyzer does not know how to extract the structure of a workflow written in DSL2.
    - The address given is a folder, we suppose then that the workflow is written in DSL2 and that all the nextflow files in the folder compose the workflow, we analyze these files to be able to extract the information on the processes. These are the steps to extract the informations of the processes :
        - Analyze the workflow
            - Create an object of type TypeMainDSL2 and initialize it (extract the processes)

File

The file object has 3 attributs :

- the address of the file
- its string (so what it contains)
- list of comments (this attribute is not used at all right now but it could be useful to save the comments in the future)

When initializing a file, these are the steps :

- Setting the string of the file to the attribute by reading the contents of the file
- Removing the comments of the string (so of the code). In Nextflow they are 2 ways to define a comment : either a comment one line *'//comment'* or multi line *'/*comments*/'*. These are the different steps to remove the comments :
    - First start by removing the scripts defined in the processes (since leaving the script part can lead to ambiguities and errors), they will put back later on, there are 2 different ways to define a script :
        - """ Script """
        - ''' Script '''
    - Remove the strings with ambiguities. These are strings with *'//'*, *'/*'* or *'*/'* present, they will be placed back later on.
    - Remove the multi line comments from the string.
    - Remove the single line comments from the string
    - Place the strings and the scripts back which were removed

IMPORTANT : the remove comments method is not perfect, examples in which some comments were remaining have been noticed. This is because we do not want to remove scripts or strings which wouldn't be removed, so the method isn't massively strict. But the remaining comments do not cause a problem in the extraction of the information that will be extracted from the workflow.

TypeMain

The typeMain has 2 attributes :
- processes which is a list containing the processes found in the workflow
- analyze_processes which is a boolean set to True so that the analyzer can analyze the different processes (for example extracting the different tools used in a process)

When initializing a typeMain, these are the steps :
- Since TypeMain inherits from File, the initialisation of the file is undergone

TypeMainDSL1

The typeMainDSL1 has 8 attributes :
- functions which is a list containing the functions found in the workflow
- channels which is a list containing the channels found in the workflow
- added_operators which is a list containing the added operators found in the workflow
- can_analyze which is a boolean set to True so that the analyzer can analyze the workflow
- name_workflow which is a string which will contain the name of the workflow later on
- nb_edges which is an int, it will count the number of edges in the structure
- nb_nodes_process which is an int, it will count the number of processes (node processes) in the structure
- nb_nodes_operation which is an int, it will count the number of operations (node operation) in the structure

When initializing a typeMainDSL1, these are the steps :
- Step 1 : Since TypeMainDSL1 inherits from TypeMain, the initialisation of the typeMain is undergone
- If there is the same amount of open courlies than closing curlies :
  - Step 2 : Finds and adds the processes to the list of processes and analyses them (see explanation Process)
  - Step 3 : Finds and initializes the functions, then formats them
  - Step 4 : Replace the condition *‘a = c1 ? val1 : val2’* by *‘if(c1){ a=val1 } else{ a=val2 }’* since channels can defined in this way
  - Step 5 : Finds and analyzes every operation, and adds them to the list of operations
  - Step 6 : Reconstructs the structure of the workflow from the information that has been extracted
  - Step 7 : Save the information extracted

TypeMainDSL2

When analyzing the object of type TypeMainDSL2 these are the steps :
- Retrieve all the nextflow (with the ‘.nf’ extension) files in the address given
- For each file found :
  - Create an object of type TypeMain
  - Initialize it

    - Retrieve and analyze the processes in the file
    - Add the processes to the list of processes of the workflow
    - Save the informations of the processes into a json

Function

The function object has 2 attributs :

- its string (so what it contains)
- its name

When initializing a function, these are the steps :

- Extract the name of the function

IMPORTANT : In DSL1 functions are not important to the structure of the workflow, since they mostly do auxiliary tasks. Therefore they are 'removed' (formated) to simplify the analysis later. If the analysis of the functions needs to be done, the corresponding methods should be added to this class.

Operation

When initializing an operation, the method extracts the channels which the operation has as origin and gives, these can be seen as sorts of inputs and outputs. We don't use this vocabulary to not cause confusion with the processes. The origin and gives are important for the structure hence they are extracted here.

Explanation Process

Algorithme d'Extraction des Informations dans les process

Pour être sûr d'analyser correctement un Process, il faut qu'il suive le modèle suivant sinon on risque de manquer des informations ou de mal les classer.

Tous les processus doivent commencer par une liste de directives. Puis pour les autres, les blocs doivent commencer par le mot clé et un retour à la ligne suivi des informations du bloc).

```
process < name > {

   [ directives ]

   input:
    < process inputs >

   output:
    < process outputs >

   when:
    < condition >

   [script|shell|exec]:
   < user script to be executed >

}
```

Explication de l’Algorithme

L’algorithme d’extraction des différentes parties d’un Processus prend en paramètre d’entrée le String d’un processus (du mot process à la dernière accolade).

Cet algorithme part du string “complet” du processus puis à chaque étape on met à jour ce string en enlevant le bloc du processus analysé. A la fin de l’extraction, on vérifie que ce string est vide. S’il est vide alors on a analysé toutes les informations sinon il y a des parties que nous n’avons pas analysées et nous n'ajoutons pas le workflow d’où provient ce processus dans nos futures analyses. Une raison pour refuser un processus est la présence de deux fois le même mot-clé (deux fois le mot-clé inputs par exemple).

Étape 1 : Préparation du String
Pour éviter les "faux-positifs scripts", on modifie le string passé en argument. Pour cela, on recherche un pattern de script (annexes) et on modifie le début du string jusqu’à ce pattern. Si nous sommes en présence d’accolades ou de parenthèses sur plusieurs lignes, le programme modifie le string pour mettre sur une unique même ligne.

Étape 2 : Extraction du nom du Processus
On extrait le nom qui se trouve entre le mot process et la première accolade au tout début du script.

Étape 3 : Extraction des Directives
Les directives se trouvent au début d’un processus et se terminent dès le premier mot-clé d’une autre partie.
Après avoir extrait cette partie, on sépare le string obtenu en une liste de directives.

Étape 4 : Extraction des Inputs
Dans un processus, les inputs sont situés après le mot-clé “input:” et jusqu’au prochain mot-clé.
Analyse du string input: séparation des inputs. Puis pour chacun extraire le type de chaque input et le nom du channel faisant le lien entre tous les processus dans le workflow.

Extraction du nom :
Présence du mot From : on extrait tout le string après ce mot.
- S’il n’y a qu’un seul mot après le mot from, cas simple: on extrait ce mot.
- Si après le mot from, il y a un channel, on l’analyse et on extrait ses origines. Si ceux-ci sont des pointeurs on les extrait.
- S’il y a un mot “simple” (sans point) suivi de parenthèses avec des channels à l’intérieur, on est en présence d’une fonction et on récupère tous ses paramètres et on les analyse un à un (mot simple ou channel).

Absence du mot From: on extrait le nom juste après le qualifier. Dans le cas d’un tuple, on extrait tous les noms de chaque type.

En sortie, on obtient un tableau contenant pour chacun un indice correspondant à un input donné et son nom général pour faire le lien dans le workflow.

Étape 5 : Extraction des Outputs
Dans un processus, les outputs sont situés après le mot-clé “output:” et jusqu’au prochain mot-clé.
Analyse du string output: séparation des outputs. Puis pour chacun extraire le type de chaque output et extraire le nom du channel faisant le lien entre tous les processus dans le workflow.

Extraction du nom :
Présence du mot Into : on extrait le nom juste après le qualifier:
- S’il y a un seul mot après le mot into, cas simple: on extrait ce mot.
- S’il y a une liste de channels séparée par des virgules après le mot into, on les sépare et on les extrait un par un.

Absence du mot Into : si le nom du channel n’est pas explicitement écrit, on extrait le nom juste après le qualifier.

En sortie, nous obtenons un tableau contenant pour chacun un indice correspondant à un output donné et son nom général pour faire le lien dans le workflow.

Dans les workflows de type DSL 2, les noms globaux sont donnés après le mot “emit:” au lieu de ‘into’.

Étape 6 : Extraction des conditions When
Pour cette partie, on extrait du string “when:” jusqu’au prochain mot-clé.

Étape 7 : Extraction de la partie Stub
La partie Stub commence au mot-clé stub et finit à la fin du processus ou bien à un prochain mot-clé s’il n’est pas placé en dernière position.
Cette partie comprend les commandes subsidiaires. On analyse en premier le langage et si les commandes sont écrites en bash alors on extrait les outils et les annotations associés.

Étape 8 : Extraction de la partie Script
La partie script peut être définie par plusieurs patterns (annexes). C’est pour cette raison qu’on l’extrait en dernier, le string “a été nettoyé” des autres parties et il y a une probabilité faible de tomber sur un faux positif. Une fois le script extrait, on analyse le langage et si celui est écrit en bash, on extrait les outils et les annotations associés.

Analyze Operations to extract Channels

In TypeMainDSL1, the initialization method calls *‘extract_analyse_operations’* to extract and analyze the operations. Before we start, let’s define what an operation is.

An operation is the concatenation of nextflow operators to create or manipulate channels; they can be seen as more fundamental and simpler processors. Nextflow defines operators as methods that allow you to connect channels to each other or to transform values emitted by a channel applying some user provided rules. See here for more details.

These are the different steps of the *'extract_analyse_operations'* method :

- Start by testing the different methods to extract the information from the operations with the *'tests'* method in the 'Operation' class, to make sure these methods are working as expected (this method is also good to get an idea on how the different extraction methods work)
- Extract the added operators created with branch or multimap, and add *'()'* at the butt when they are used the workflow (this is to simply their identification later on)
- Extract the operations which start by '*Channel*' or '*channel*', add them to the list of operations and format them.
- Extract the operations which use the operators (we start looking at the first one and move our way down), add them to the list of operations and format them.
- Extract the case where there is '*channel = OPERATION_X*' where '*OPERATION_X*' is an operation that has already been extracted and formatted
- IMPORTANT : Normally in nextflow DSL1, once a channel is defined it can not be modified, since a channel can be defined in 2 ways (https://www.nextflow.io/docs/latest/channel.html) :
    - A queue channel is a non-blocking unidirectional FIFO queue which connects two processes or operators
    - A value channel e.g a singleton channel by definition is bound to a single value and it can be read unlimited times without consuming its content.

  With that in mind, during the development of the analyzer, we ran into multiple examples where we would modify a channel as such : '*ch = ch.mix*'. With this occurring multiple times in the workflow. Testing this, it does execute and work. Here is an example on how this work, imagine a user writes :

  ch = fromFile(..)
  ch = ch.mix(a)
  ch = ch.mix(b)

  This is equivalent to :

  ch = fromFile(...).mix(a, b)

  So the analyzer has to take this into account, that is what the method 'find_problematic_operations' does, and this is the next step.
- Initialize the operations, this means retrieving the origin and the gives (the channels) of all the operations which have been found.
- The last part is to find the occurrences of '*variable1 = variable2*' because '*variable2*' could be a channel, in that case so is '*variable1*', this is why we initialized the operations found above.
- Initialize the remaining operations,e.g. retrieving the origin and the gives (the channels) of all the operations which have been found.

Reconstruction of the structure

The reconstruction of the structure is one of the last steps of the analyzer, when we call this method the extraction and the analysis of the processes and the operations has already taken place. We use this information to reconstruct the structure.

These are the steps :

- Start by defining the graphviz graph
- Create 2 empty lists : tab_origin and tab_gives, every time that a node (either a process or an operation) is added to the graph the inputs of the node need to be added to tab_origin, and the outputs in tab_gives, the name of the node (either process or operation) also needs to be added to the list (more explicitly [‘key’, ‘name_node’] is added), these lists are to be used later on
- Add all the processes to the graph as process nodes, and add their corresponding inputs to tab_origin and outputs to tab_gives
- Define an empty list list links_added which allows us to check if we have already added a link (edge/ channel) to the graph
- For each process p1 :
    - Check if the output of p1 is the same as the input of another process p2, so that p1.output == p2.input (this means it’s the same channel). If so, add the edge to the graph
- While an edge has been added to the graph do :
    - For every element in tab_origin :
        - Check to see if the gives of an operation can be linked with the origin, if the node of the operations has not been added to the graph, add it. Also add the edge to the graph to represent the link
        - Check to see if the output of a process can be linked the with the origin, if so add the edge to the graph to represent the link
    - Add the newly added operation to the graph to tab_origin and tab_gives
    - For every element in tab_gives :
        - Check to see if the origin of an operation can be linked with the gives, if the node of the operations has not been added to the graph, add it. Also add the edge to the graph to represent the link
        - Check to see if the input of a process can be linked the with the gives, if so add the edge to the graph to represent the link
    - Add the newly added operation to the graph to tab_origin and tab_gives
- Save the graph

Dependencies

- setuptools
- argparse
- pathlib
- os
- glob2
- re
- graphviz
- json
- rdflib
- jellyfish
- glob
- threading
- queue
- time
- sys
- functools
- ntpath