

Supplementary material for CAISE 23 - RPA testing using symbolic execution

Ciprian Paduraru, Marina Cernat, Adelina Staicu, and Alin Stefanescu

University of Bucharest, Computer Science Department

1 Applications set

1.1 Application to detect the type of a bank loan

The first application is named BankLoan and its purpose it's to verify the type of a loan based on its amount and period of time. If the loan is smaller than 1000, then it is considered a low volume loan. If the amount is between 1000 and 100000, then the loan has a medium volume and if it's over 100000, then the volume is considered high. In case of a high volume loan, then a pin should be checked. The workflow for checking the pin is meant to verify if the robot can provide a valid pin for the bank account and it also has a retry mechanism (for example, the expected_pin can be "1234"). If the robot can not provide a valid pin for the bank account, after a certain number of retries, the workflow fails. All types of loans also have a period of time associated. If the term for a loan is less than five years, then it is considered a short-term loan, otherwise it is a high term loan.

For this model, a test case was created. By running the test case, some testing data will be generated and in this way a full coverage of the model activities will be obtained.

In the given section, the data for testing the workflow will be generated, by invoking the GenerateTestData.xaml workflow. In this workflow, a configuration file is read, file which contains the following details: the paths for the python script that connects all the external components involved in testing the workflow, the path for the working directory and the python home path along with the queue name from the Orchestrator (which in this model will not be used). After the configuration file is read, the python script that runs all the external components is called and the test data is generated. In the when section, the generated test data is written into csv files, from where it will be taken in the execution of the workflows. In the then section, the main workflow will be invoked. In order for the data to be generated, the fuzzer component needs some constraints for each variable. In this way, the fuzzer will give to the robot data that belong to a certain interval, it can be passed through a regular expression or it can have a special type (like true or false). To create these constraints which help the fuzzer to generate data, the RPA developer can add some annotations in the workflows. These annotations can be added at the workflow level, or at the activity level. In the BankLoan example, annotations can be found in different places of the workflows, but we will describe two scenarios in which we needed

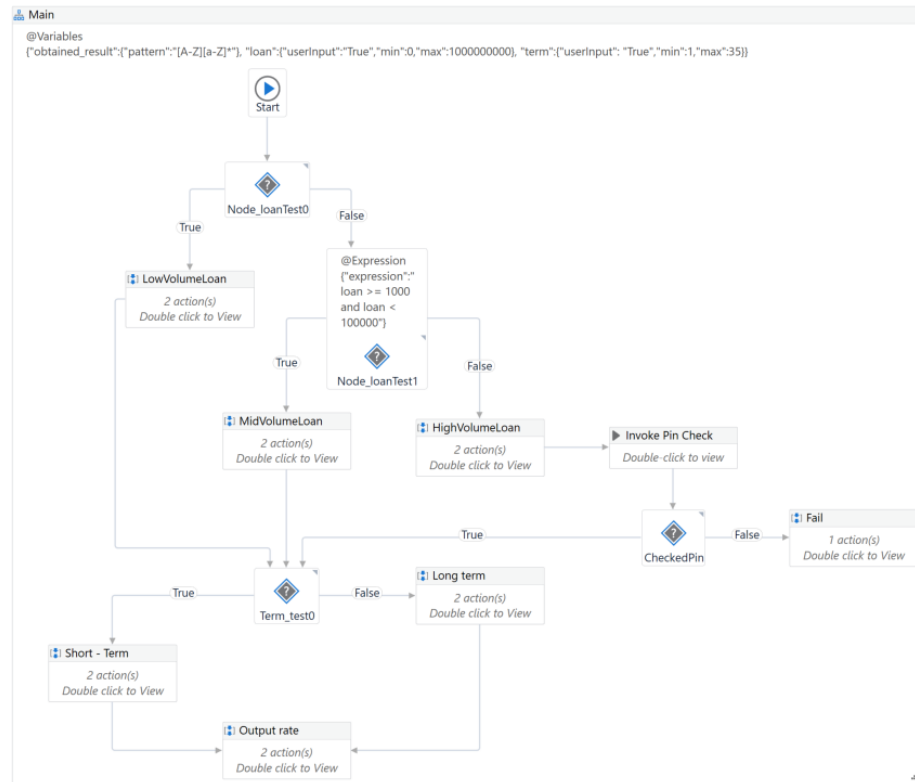


Fig. 1. The main workflow of BankLoan application

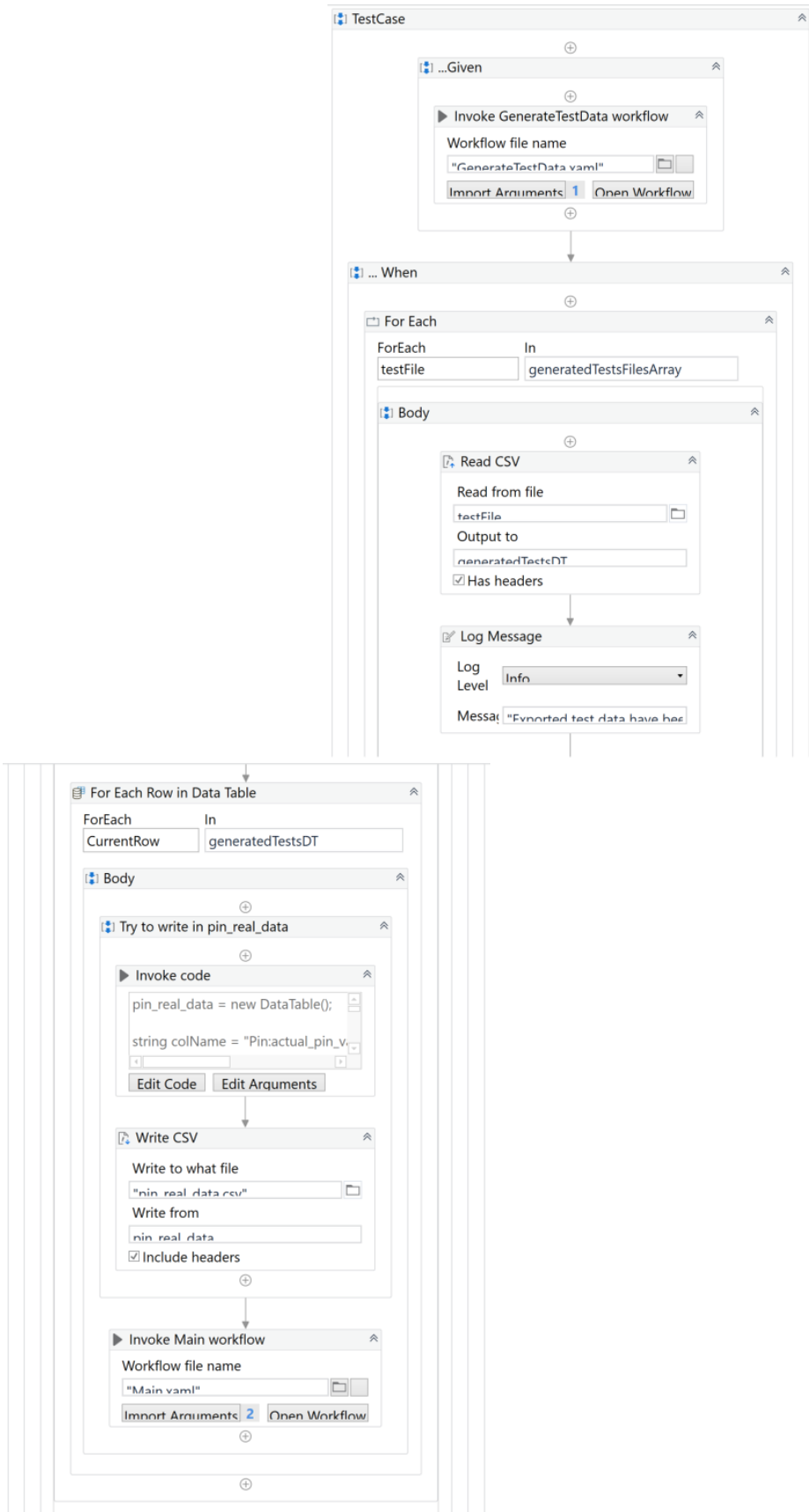


Fig. 2. Test case for bank loan application

to add them. The first scenario will be in the Main.xaml workflow, where an annotation is used for the main variables. In this annotation a range is specified for each variable. Based on that range, the fuzzer component can then generate values which belong to the specified interval. The loan variable can take values between 0 and 1000000000 and the term variable can take values between 1 and 35. The second scenario is in the pin workflow, where an annotation is used to restrict the data generation for the number of retries when the robot enters the pin to access the bank account or for the pin values. Here, the number of retries can be between 0 and 3 and the values of the pin can be between 1111 and 9999. After using the annotations, the generated data is written into a csv file named generatedTests which will be processed into the testing sequence. The data extracted after this process will be given as parameters to the workflows when they are invoked.

1.2 Model that automatizes the loaning operations for a banking process

Similar with the previous model, but this time with some user interface interactions, this model navigates to "<https://uibank.uipath.com/>" to check if a person is qualified or not for a loan with UiBank - the prototype website of a bank created by the people from UiPath academy to better illustrate their examples. After navigating to the UiBank main page, the model clicks the "Products" button from the upper right corner of the page and then selects "Loans" from the drop-down menu. On the loans page, the robot clicks on "Apply for a Loan" button. On the next page, some values are inserted, like: the email address of the requester, the loan amount requested, the loan term, the currently year income of the person applying for the loan and it's age. If all these data fall within a certain interval, for example if the value of the current year income is large enough and in accordance with the value of the loan or the age of the requester is at least 18 years old, then the person is qualified to request a loan to UiBank. If one of this conditions is not met then the the person who wants to access a loan is not qualified and must wait until all the criteria are satisfied. The test data that can be generated through a test case for this model are the amount of the loan and the fact that the user is or it's not qualified.

1.3 Application for employees management

In this application, certain actions can be done depending on the type of the user. The workflow interrogates a table of employees which stores data like: name, salary, register date, if the employee has a master degree or not, contract type, the name of the project, leave request and notice period. If the user is an administrator, it is allowed to update the salaries for the employees who have the register date greater than one year and a master degree. If the user has an HR role, then it can verify the leave requests for the employees. The leave requests are not approved if a person with a short term contract asks for unpaid leave or the notice period is smaller than five days. The workflow also calculates the

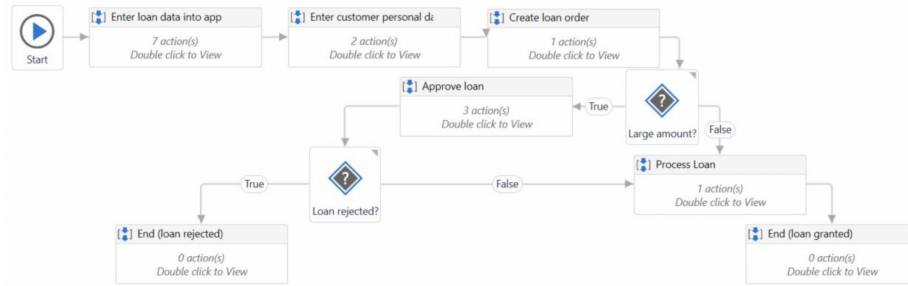


Fig. 3. An example of RPA workflow that automatizes the loaning operations for a banking process.

number of employees who have a master degree and the number of employees per project. If the number of people who have a master degree is smaller or equal with two, then the employer should encourage people to study more and if the number of people who work on a project is smaller than two, then more people should be assigned to that project.

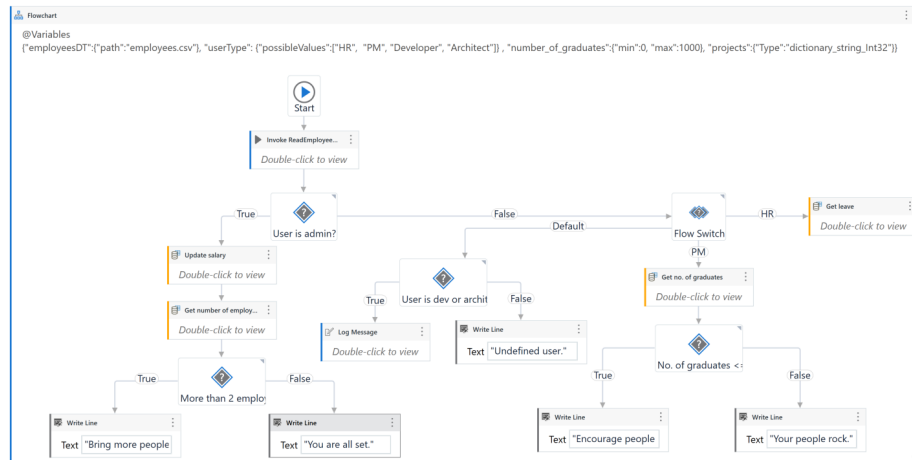


Fig. 4. The main workflow of employee application

A test case for this model generates test data that offers full coverage for the workflow. The generated data are the type of the users, the number of graduates and the number of employees assigned on a project.

1.4 Application which lets you choose from the services of a hospital

PrivateHospital is a model which lets the user choose some medical services and calculates the final price, adding some discounts to some services, where necessary. When it runs, after a welcome message, the robot asks the user to introduce its age. A list of medical services is displayed and the user can choose the desired service. After a medical service is picked from the list, the robot displays the price of that service and asks the user if he wants to add another one. If the user wants to add another service, the process is restarted and the price of the next chosen service will be added to the final cost. If the user decides to stop adding medical services, the final price is displayed and the process ends. At the beginning of the process, the user is asked to introduce its age in order for some discounts to be applied at certain services, depending on the range it belongs to. If the user's age is less than 18 years, then the user has a 50% discount at some services. If the user's age is greater than 65 years, then the user has a 60% discount at some services and if the user is between 18 and 65 years old, then it doesn't receive any discounts. The data that can be generated from a test case is the age of the user, the response of the user input that is extracted from an input dialog (it can consists in one of the three services taken as an example: MRI, Neurosurgery consult or Physical therapy) and the choice of adding or not another service.

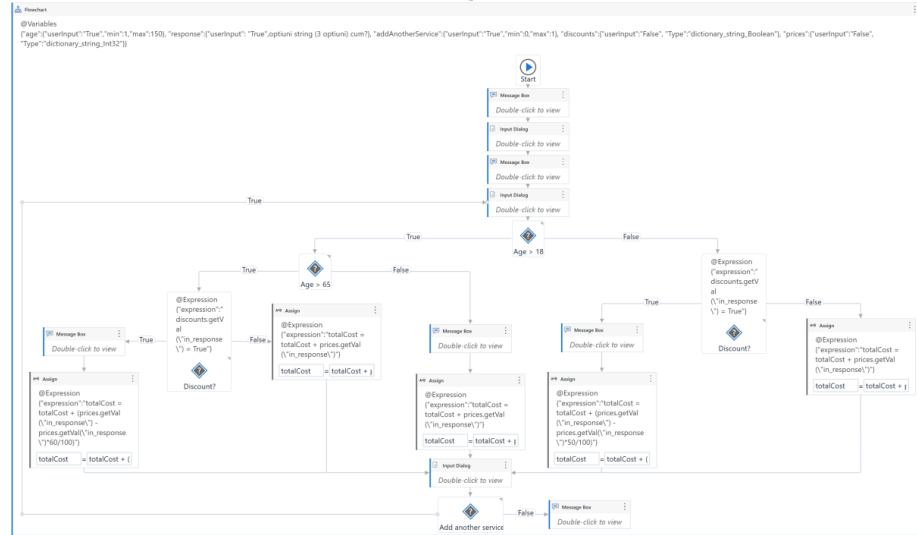


Fig. 5. The main workflow of hospital application

1.5 InvoicesProcessing - A producer-consumer communication pattern

Producer and Consumer are two separate models that interact with each other, processing elements added on a queue in the Orchestrator, a web application that allows you to orchestrate the execution of repetitive business processes by creating, monitoring, scheduling, and controlling the automated bots and processes. The producer searches for all the PDF files saved into a folder present locally on the user's machine and analyses them. Those PDF files are actually invoices from which the model extracts data like vendor and invoice number and adds that data into a queue in the Orchestrator. The consumer will read those elements added into the queue, until no more elements can be found, and will process them and write everything into an excel file. For this two processes, a test case can generate test data on the consumer side, for the flag that is coming from Orchestrator. The Orchestrator gives a flag named "ShouldStop" which has a Boolean value. When "ShouldStop" is on true, it means that there are no more transactions in the queue, and the consumer workflow can stop processing data from Orchestrator. If "ShouldStop" is on false, then some other data is present in the queue and the consumer should continue to process that data. A test case created for the consumer model can generate test data for the "ShouldStop" flag given by the Orchestrator.

1.6 InvoicesManagementApp - a wider and more complex producer-consumer paradigm

Producer and Consumer are two separate models that interact with each other, processing elements added on a queue in the Orchestrator, a web application that allows you to orchestrate the execution of repetitive business processes by creating, monitoring, scheduling, and controlling the automated bots and processes. The producer searches for all the PDF files saved into a folder present locally on the user's machine and analyses them. Those PDF files are actually invoices from which the model extracts data like: client, invoice number, due date, unit price, discount, amount and VAT number and adds that data into a queue in the Orchestrator. For the VAT number, the robot checks the country code from the invoice, in order to add the right value of VAT. If the payment due is in more than 15 days, then the invoice values are not added into the queue in Orchestrator. Also before adding the data into the queue, the robot verifies into a database if the client has any unpaid invoices. If it has unpaid invoices, the robot will also add into the queue the value of each unpaid invoice along with a penalty of 0.3% for each day of overdue. The consumer will read those elements added into the queue, until no more elements can be found, and will process them and write everything into an excel file. For this two processes, a test case can generate test data on the consumer side, for the flag that is coming from Orchestrator. The Orchestrator gives a flag named "ShouldStop" which has a Boolean value. When "ShouldStop" is on true, it means that the robot was stopped by an user in the Orchestrator and the consumer should not

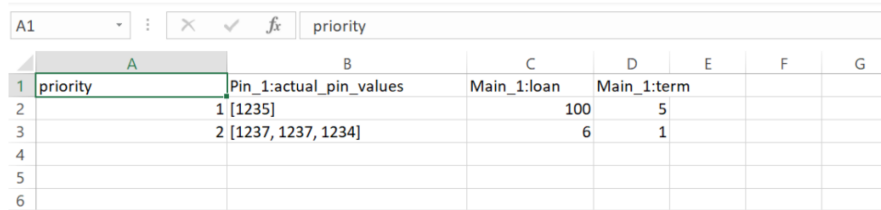
continue processing items into the queue. The robot will continue to process the elements from queue until all of them are consumed. A test case generated for the producer model can generate data for the payment due, if the vendor has unpaid invoices or not, the number of overdue days and the country code. And a test case created for the consumer model can generate test data for the "ShouldStop" flag given by the Orchestrator, the number of transactions left into the queue and also for the fields extracted from invoices: vendor, invoice number, due date, unit price, discount and amount.

1.7 SpreadsheetsStateMachine - A state machine data processing paradigm

The SpreadsheetsStateMachine is a model which reads an excel file which contains details about some employees, filters the data by the salary column and writes that filtered data into a new excel file. In that new excel file should be present only the employees that have the salary smaller than 10000. This model is built using state machines, as its name suggests.

2 Framework details

2.1 Seeds injection



	A	B	C	D	E	F	G
1	priority	Pin_1:actual_pin_values	Main_1:loan	Main_1:term			
2	1	[1235]	100	5			
3	2	[1237, 1237, 1234]	6	1			
4							
5							
6							

Fig. 6. An example of inputs seeds injection in the process of testing the RPA workflow for a banking loan process, using concolic execution. The priority column designates what inputs are more important to try first.

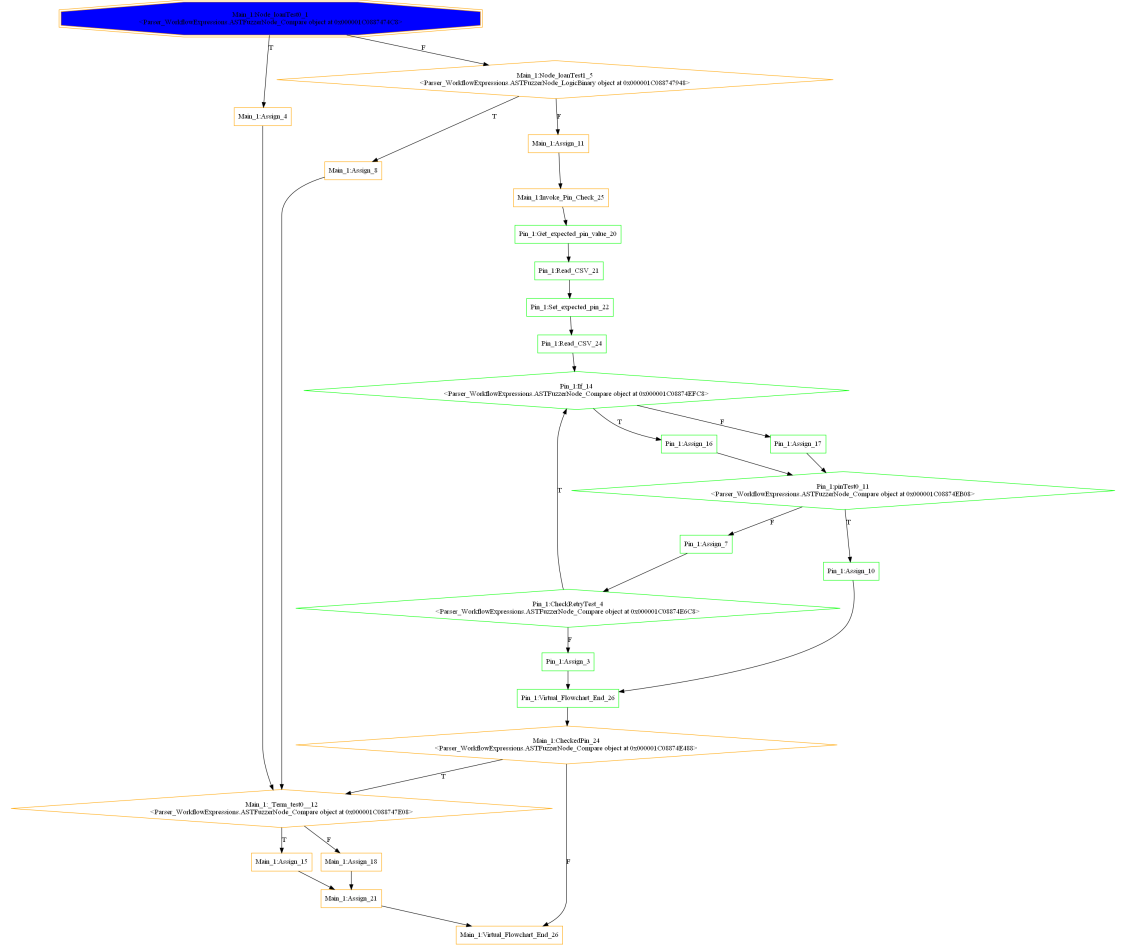


Fig. 7. An example showing how an RPA workflow in a banking loan use-case is translated in our framework to a graph of nodes, and their corresponding branches for the BankLoan application.

```

"Pin_1:If_14": {
  "expression": "actual_pin_values.ElementAt(local_number_retries) == expected_pin",
  "Annotation": {
    "expression": "actual_pin_values.GetElementAt(local_number_retries) == expected_pin"
  },
  "transitions": [
    {
      "value": "True",
      "destination": "Pin_1:Assign_16"
    },
    {
      "value": "False",
      "destination": "Pin_1:Assign_17"
    }
  ]
},

```

Fig. 8. Annotation at the code level, example from our source code repository. Basically it helps translating a piece of original source code to something easier for the fuzzer. This is also the pattern used for mocking API calls that need to be hidden at test time for either performance or proper results evaluation.

Output:

```

"variables": {
  "out_pin_check_successful": {
    "Type": "Boolean"
  },
  "local_number_retries": {
    "Type": "Int32",
    "Default": "0",
    "Annotation": {
      "min": 0,
      "max": 3
    }
  },
  "local_pin_test": {
    "Type": "Boolean"
  },
  "expected_pin": {
    "Type": "Int32"
  },
  "actual_pin_values": {
    "Type": "Int32[]",
    "Annotation": {
      "bounds": 10,
      "min": 0,
      "max": 9999,
      "userInput": "True"
    }
  },
  "local_test_data_expected": {
    "Type": "DataTable",
    "Annotation": {
      "path": "pin_mocked_data.csv"
    }
  },
  "local_test_data_actual": {
    "Type": "DataTable",
    "Annotation": {
      "path": "pin_real_data.csv"
    }
  }
},

```

Fig. 9. Annotation at the code level, example from our source code repository. Currently, we support specification for types, boundaries for both individual values and array lengths, and patterns for string variables.