QUESTIONS AND ANSWERS TO THE SDEVAL PROJECT

ALBERT HEINLE

Abstract. This document deals with some common questions that may arise when using SDEval. If you have other questions, please contact us.

For the terminology used in this document you might consider the file `definitions.pdf` in the same directory.

1. Questions concerning how to add things to SDEval

Q: My favourite computer algebra system can solve some of those computation problems you are already dealing with. Why is it not there as an option? And how can I help you put it in there? **A:** We are sorry and happy in the same moment. Sorry, because your favourite computer algebra system is not yet in the list of SDEval, and happy because you are a person who has knowledge about another computer algebra system that we might add to our list of computer algebra systems.

The reason why it is not there yet is either that we did not know about this system, or we do not know how to perform certain computations using this computer algebra system. Clearly, the first option implies the second.

So how you can help us. There are two ways.

(1) You have a general knowledge about Python. That is the best case. Then go to the folder `classes/templates/comp`. There, you can see folders representing the different computation problems that are currently available within SDEval. Those names come from entries in the Symbolic Data Table `COMP`. Find the one that your favourite computer algebra system can solve, and enter the folder. There you see a bunch of folders named by computer algebra systems. Add a folder there named by your favourite computer algebra system. In this folder, you create a file called `template.py` and a file `__init__.py`. The content of the file `template.py` can be copied from the template of another computer algebra system. There, you can find a method called `generateCode`. Do not change the arguments. Using those arguments, let this method create a string with executable code and return it in the end. After that, nothing more needs to be done. Test this function, and then send it to us. Thank you in advance.

(2) You have no idea of Python. Then just send us an example code how to solve an already stated computation problem in your computer algebra system and we will do the work for you.

Q: I often have a computation problem that is not listed in your choice of computation problems. How can I add this, so that I can do automated testing and benchmarking in the future on that? **A:** That is a good thing.

We always want to expand our software as much as we can. The best thing to do is to talk about it with the Symbolic Data team in order to add this problem in the `COMP`-table of the project and in order to have a clear description of it.

Meanwhile, we can test-wise add it to our system. Just provide us with a good description of the problem, what kind of inputs might be needed and what computer algebra systems you know are available out there. Furthermore, you need to send us some test code.

If you are familiar with python, you can even make it test-wise yourself. Afterwards, we would only love to have the code you produced in order to may add it to the project.

Precondition: You know an SD-Table with problem instances where the inputs for your computation problems can be derived from. If that is not given, see next question.

How you do it: in the folder `classes/comp/`, add a folder (with an `__init__.py`). The name shall represent in some way your computation problem. Then, like in the other computation problems, make subfolders with possible computer algebra systems, and add the templates for them as done for the other computation problems. See the previous question.

Once you have done that, we need a representation of your computation problem as a class. For that, go into the folder `classes/computationproblems`, and as for the other computation problems there, you create a file containing a class representing your computation problem. Most of it can be copy and pasted from the other computation problems. The important thing is only to add at least one associated SD-Table, where instances can be taken from for this computation.

The last step is to "register" this computation problem. For that, open the file `classes/TaskFolderCreator.py`. In the end of the method create, you find a case differentiation on different existing computation problems. Choose one that is using the same SD-Table as your computation problem, and copy the line to the case of your computation problem, that you add. That is it. But of course, it takes a bit to read through it. The fastest way is just to contact us.

Q: I have created/need a new table with problem instances. How do I add it?
A: In general, this is quite complicated to add. Please contact the SymbolicData team, try to explain to them why this table might be relevant, and how you can help them to add it to the project. Once it is in and we can derive inputs from it for our computation problems, we will add it to our project eventually. Feel welcome to always give us hints how we can use recently added SD-Tables.

2. Questions on using SDEval

Q: I have created an task folder. Now I want to change some parameters, add some lines of code, etc.. To what extend is that possible without making the task folder not working any more when running `runTasks.py`? **A:** There are different things you can do. If you just change lines in the executable files for the computer algebra systems, nothing major can happen. Feel free to do so.

If you want to add new files, for example for other computer algebra systems: Be careful, but it is possible. Just create in the same way as for the other computer algebra systems a folder containing a file called `executablefile.sdc` and named by your computer algebra system (for EVERY problem instance. This file may be empty, but it must be there). Then, in the `machinesettings.xml`, add your computer algebra system to the list and the command to execute it on your target machine. Last, but not least, also add it in the `taskInfo.xml` file.

In the same way, you can also add by hand another problem instance. But again, make sure that you also have executable files for EVERY computer algebra system. They may be empty, but they must be there.

Q: I have already created an export task folder. Now, on my target machine, the commands to call a certain computer algebra system have changed. How can I adjust that without creating the same task again? **A:** That is no problem. Just go into the `machinesettings.xml` in the task folder and change it there. It is easy to find.

Q: I start the `create_tasks_gui.py`, and somehow he is constantly asking me to open a folder. What is going on there? **A:** You might have moved SDEval out of the Symbolic-Data folders. Fact is, that the files of Symbolic Data must be on your machine. Just search for the folder `XMLRessources` in the Symbolic-Data folders, and select it in the popping up window.

Q: I just want to run some experimental calculations without adding it as a computation problem on its own to symbolic-data, so the task generation is not that interesting for me. But I like your `runTasks` routine. Can I also make an folder on my own and run the calculations with the profiling provided by this method? **A:** It is not meant to be used like that, but well, of course you can do it. Just do the following steps:

(1) Create a folder and copy the file `runTasks.py` and the folder `classes` to it.
(2) Within the created folder in the previous step, create another folder called `casSources`. In this folder, make subfolders representing the different calculations you want to do. Within those folders, create subfolders named by the programs you want to use to do that. Every calculation shall contain the same programs. Finally, in those folders that are named after programs, write a file `executablefile.sdc` containing the executable code for the programs.
(3) Within the folder created in the first step, add files named
    `machinesettings.xml` and
    `taskInfo.xml`. You need to fill them with data. How these XML-files should look like is described in the "For Developers" section of the documentation (in the `MachineSettings` module resp. the `TaskToXMLWriter` module). Alternatively you can just build a random taskfolder with the `create_tasks[_gui].py` and derive the format of those files from the produced folder.

Q: In the Taskfolder, you have the folder `casSources` and a file called `taskInfo.xml`. Why do you need the additional `taskInfo.xml` file, when you also could just read the files in `casSources` and derive the corresponding computer algebra systems from the folder names? **A:** Some computation

problems take longer than others. If you only wish to make a quick check if e.g. after a change in your code everything is still working properly, you first consider the tasks that are known to be done very fast. Therefore, you can comment the problem instances in the `taskInfo.xml` file that are known to take a while, and for the next run, they will be ignored. As you see, the additional file serves the purpose to make the use of the taskfolder more flexible to the user.