— Exercise 1 —

Deadline: 16th October

Task 1 - Module

Clone the git repository of your group. Create a new module in this repository using the script `writeModule.py`, whose only argument is the new modules name. You can execute this script with the following command in the SMT-RAT root directory:

```
python writeModules.py <name>
```

Now you should find a new folder named `<name>Module` in the folder `src/lib/modules/`.

Within this module, you will work for the rest of the practical course. Information about the members of a module, the interfaces to implement and auxiliary functions are available in the SMT-RAT Wiki [1].

In the new folder is also a tex-file, use this file to document your work (decisions about your implementation, ideas (Did they work? If not, why?), ToDo-lists, etc.). Annotate your entries with the corresponding date and mark finished tasks instead of deleting something, such that we can evaluate your progress.

Task 2 - Strategy

Next you need a new strategy to actually use your module. Therefore you create a new file in the folder `src/lib/strategies/`, you may copy one of the existing files e.g. `RatOne.h`. Change the filename to the desired name for your strategy.

Now you have to modify the content of your strategy file. The class (and constructor) have to be named identical to the file. The next thing to be changed are the includes, you need to include `../solver/Manager.h`, `../modules/SATModule/SATModule.h` and your module from `../modules/<name>Module/<name>Module.h`. Last you have to adapt the backends, you use only the *SATModule* with your module as backend.

To change the used strategy to yours, execute

```
ccmake ../
```

and change the option `SMTRAT_Strategy` to the name of your strategy. If you compile SMT-RAT now, the current version of your module will be used so you can check whether it compiles.

Task 3 - Preliminaries

Read some papers (or textbooks) about *SMT solving*, *equality logic* and *uninterpreted functions*. For orientation you may start with the papers referenced on the slides.

There are two approaches for solving these logics:

- *Eager*: Encode the problem into *propositional logic* and solve it using a SAT solver.
- *Lazy*: Abstract the problem for a SAT solver and then solve it with a theory module that has to handle only conjunctions of theory constraints.

In this practical course only the second approach is considered. Therefore methods like the *Ackermann reduction* or *Bryants reduction* can be ignored.

[1] `https://github.com/smtrat/smtrat/wiki/4---Implementing-Further-Modules`

Task 4 - Test Cases

Get a rough overview of how the algorithms that are used to solve these logics work. Create at least *five* test cases to detect possible sources of errors in the implementation. Consider special cases, which might occur during the execution of these algorithms, in your test cases.

For future use write your test cases in the SMT-LIB[2] input language for SMT solvers and save them as *smt2* files in your module folder. In this format the example problem $(a = f(x) \vee a = g(y)) \wedge x \neq y$ could be expressed as

```
; example problem
(set-logic QF_UF)
(set-option :produce-models true)
(set-info :status sat)
(declare-sort A 0)
(declare-fun f (A) Bool)
(declare-fun g (A) Bool)
(declare-fun x () A)
(declare-fun y () A)
(declare-const a Bool)
(assert
    (and
        (or
            (= a (f x))
            (= a (g y))
        )
        (not (= x y))
    )
)
(check-sat)
(get-model)
(exit)
```

Further examples for the syntax can be found here:
`http://smtlib.cs.uiowa.edu/examples.shtml`.

[2]`http://smtlib.cs.uiowa.edu/`