Homework 2

Metode formale în ingineria software 2018-2019
Formal Methods in Software Engineering 2018-2019

Deadline: Wednesday, October 31, 16:00. A pdf copy of the solution will be uploaded using the link https://www.dropbox.com/request/684v31uztLrApqTmzk7O

Exercise 1 The goal of this exercise is to download the Maude system and to have a first in-depth exploration of its capabilities. The Maude system can be downloaded from the address
`http://maude.cs.illinois.edu/w/index.php/The_Maude_System`.
Read the first three chpaters from the primer
`http://maude.cs.illinois.edu/w/index.php/Maude_2_Primer_and_Examples`
in order to get a first feeling of what can be done with it. The full description of the needed capabilities is given in the manual:
`http://maude.cs.illinois.edu/w/index.php/Maude_Manual_and_Examples`.

Include in the file to upload a snapshot that shows how the system works, using an example from the primer, on your system. The snapshot should have enough details that can indicate that it is your system.

Exercise 2 The file `prop.maude` from the address
`https://sites.google.com/site/fiicoursefmse/2018-2019/hw2018-2019/prop.maude?attredirects=0&`
include a partial description in Maude of the natural deduction system for the propositional calculus. The description is self-explained. Here is an example that shows how the rules implemented in the PROD module can be used to prove $\{P, \neg P\} \vdash$ `False`:

```
Maude> in prop.maude
Maude> search {P, ~ P} |- False =>* thm .
search in RAA1 : {P,~ P} |- False =>* thm .

Solution 1 (state 1)
states: 2  rewrites: 2 in 0ms cpu (0ms real) (42553 rewrites/second)
empty substitution

No more solutions.
states: 2  rewrites: 4 in 0ms cpu (0ms real) (38834 rewrites/second)
```

The fact that Maude system founds a solution says in fact that there is a proof tree for $\{P, \sim P\} \vdash \mathit{False}$. You may see the rule(s) applied using the `show path` command (1 is the number of the reached state):

```
Maude> show path 1 .
state 0, Thm?: {P,~ P} |- False
===[ crl {Ps,~ P} |- False => thm if {Ps} |- P => thm . ]===>
state 1, Thm?: thm
```

Only one rule is displayed, the other one being used in the evaluation of the condition. It can be displayed by searching for the condition part:

```
Maude> search {P} |- P =>* thm .
search in RAA1 : {P} |- P =>* thm .

Solution 1 (state 1)
states: 2  rewrites: 1 in 0ms cpu (0ms real) (19607 rewrites/second)
empty substitution

No more solutions.
states: 2  rewrites: 1 in 0ms cpu (0ms real) (8264 rewrites/second)
Maude> show path 1 .
state 0, Thm?: {P} |- P
===[ rl {P} |- P => thm . ]===>
state 1, Thm?: thm
```

Using these rules, it is easy to build the proof tree.
Requirements:

1. The full descriptions of the natural deduction systems is not possible because it may produce nonterminating rewritings. For instance, if you uncomment the rule

   ```
   ***(
    crl {Ps} |- Q => thm
      if {Ps, ~ Q} |- False => thm .
   ***)
   ```

   then the application must be killed:

   ```
   Maude> in prop.maude
   ==========================================
   mod PROP
   ==========================================
   search in PROP : {~ P,~ (~ P \/ ~ Q)} |- False =>* thm .
   ^C
   ^Z
   [1]+  Stopped                 ~/Software/Maude-2.7.1-osx/maude.darwin64
   ```

Find the combination of rules that produce the nonterminating rewriting.

2. Find other proof rules that added to the Maude description produce nontermination.

3. The files also includes a sketch of the proof for the De Morgan law $\{\neg(P \wedge Q)\} \vdash \neg P \vee \neg Q$ (in the implementation ~ is used for $\neg$). Find out what rules were applied and show how the proof tree can be built using these rules.

Prof. dr. Dorel Lucanu