

A quick introduction to POEM*

Janusz Malinowski and Peter Niebert

April 21, 2010

1 Introduction

POEM (Partial Order Environment of Marseille) is a modular model checking tool built to support several input languages, several analysis and solving algorithms. POEM provides reusable code for some powerful partial order based model checking algorithms.

In this tutorial, we will introduce quickly the reader to the main functions of POEM. POEM is available at <http://www.p-o-e-m.org>

2 Model checking of the knights problem

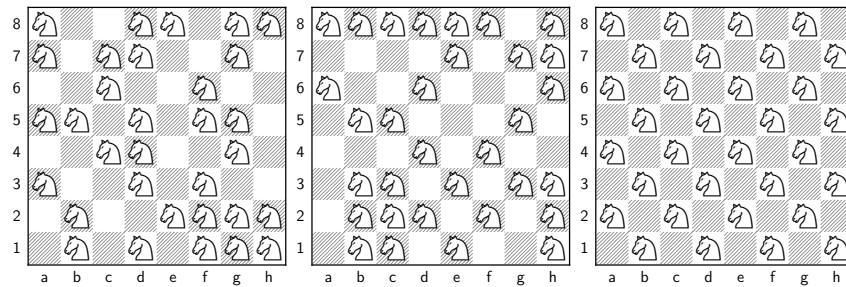


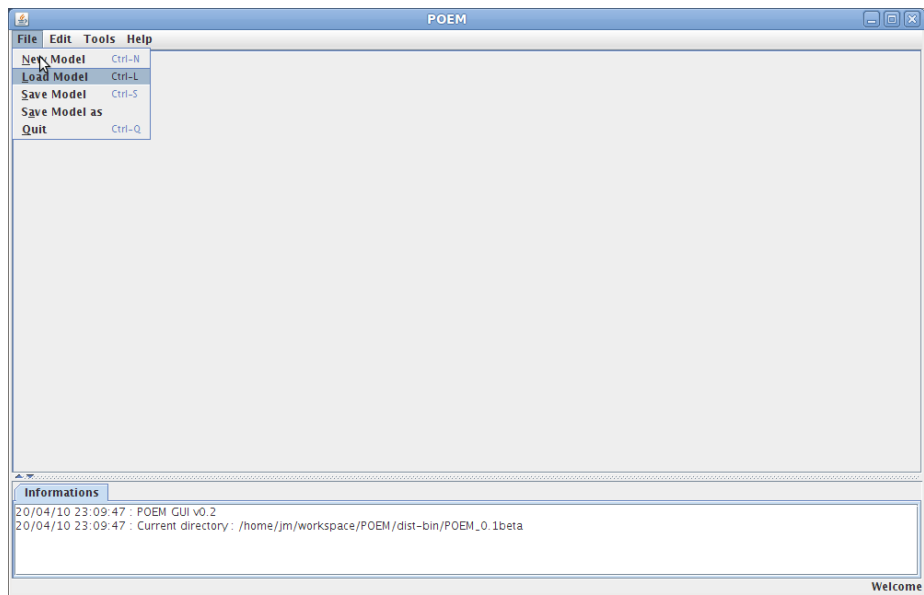
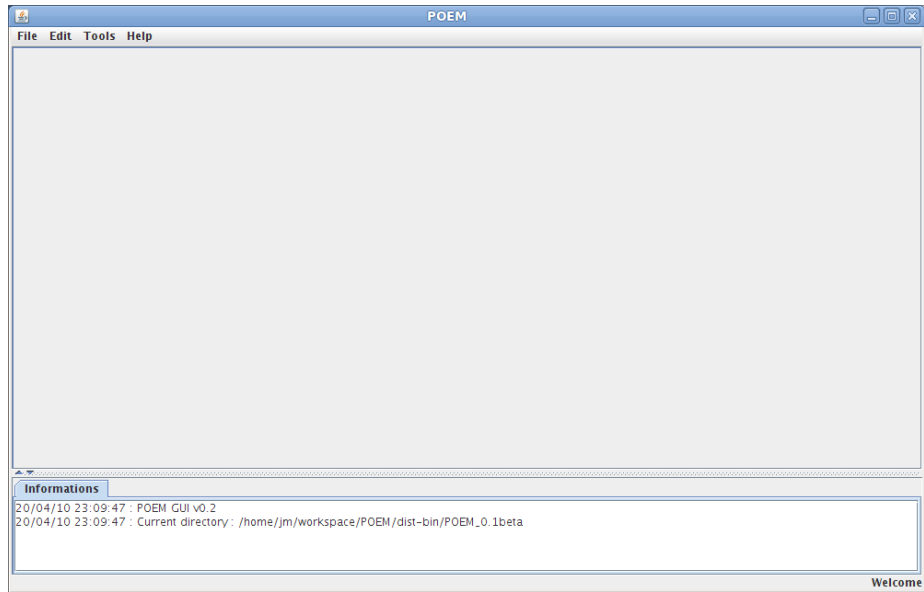
Figure 1: Solution from a random initial situation in two multisteps

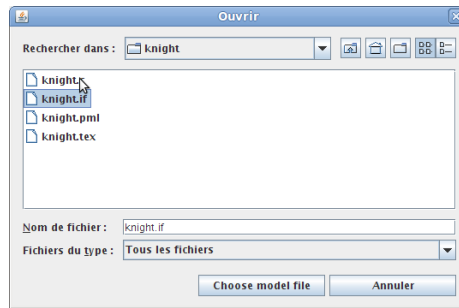
Several knights are placed on a chessboard and search a sequence of moves respecting chess rules such that in the end no two knights menace each other.

The left board in Figure 1 shows an example of an initial situation, the right board a safe position. We build a very simple model where each square of the chessboard is managed by a process which can, if a knight is on its square, move it to another authorized and empty square. A dedicated transition is used to test if all the knights are in a "pairwise safe" position.

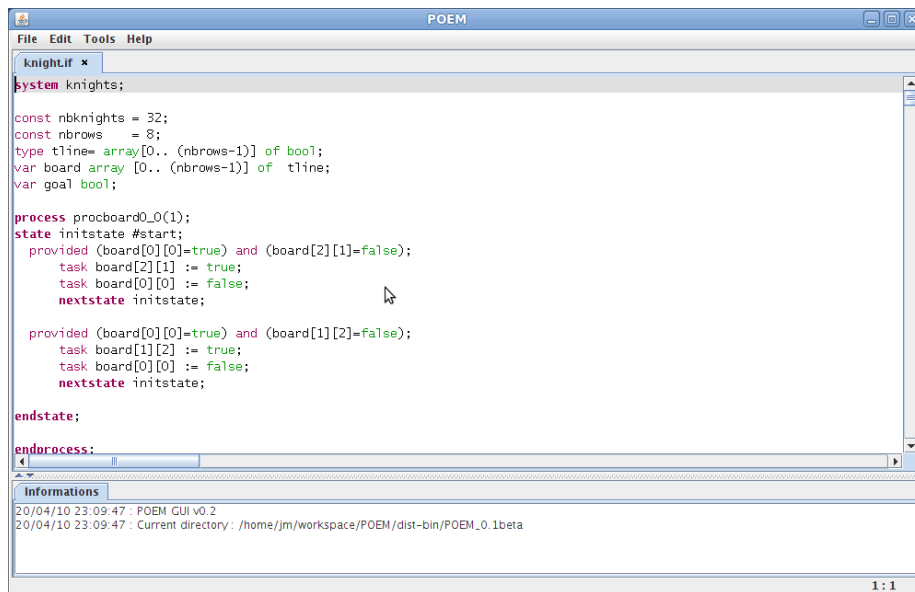
*Supported by the ANR project ECSPER(ANR JC09_472677 ECSPER).

Start the POEM Gui for your system and load the knight model



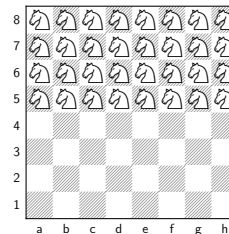


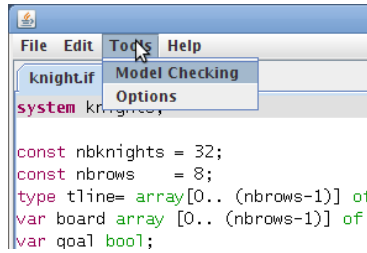
Now the model is loaded, you can have a look at it.



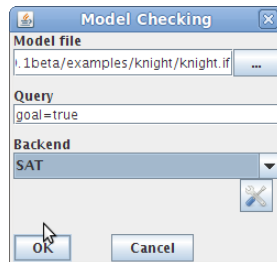
The Knights model is a very big model, because each square of the chessboard is managed by one thread. In this example, we have a 8×8 chessboard with 32 knights placed on the top part, like in figure on the right, so this model has 64 threads running in parallel.

The boolean array `board` is used to store the position of the different knights on the chessboard. A special thread tests if the condition (no knight can eat another knight) is verified and set the boolean variable `goal` to true if it is the case. So the condition we want to test is `goal=true`. In the menu `Tools`, click on `Model Checking`.

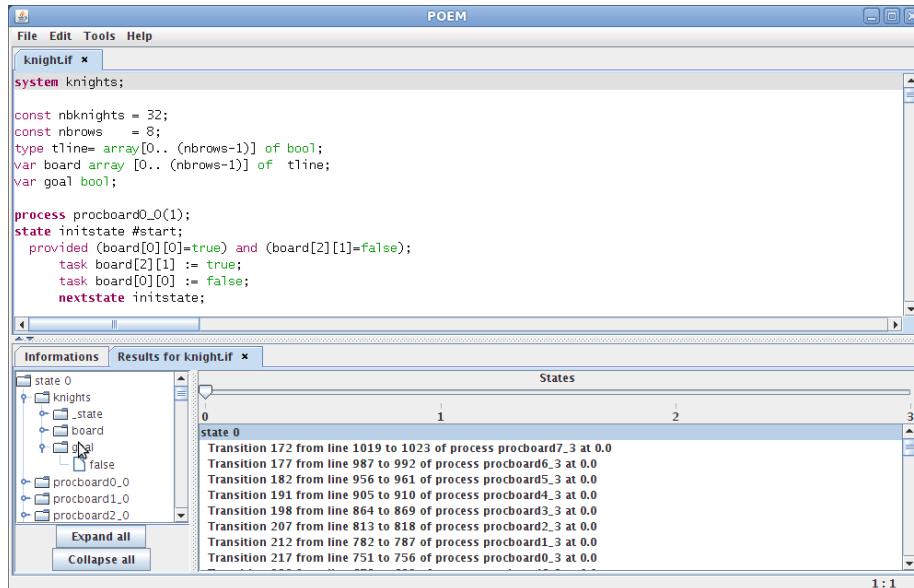




The Model Checking Window is opened and allow you to choose a file (it is done now), to enter a query (here we want to check if there exists a solution where the condition `goal=true` is verified) and a backend (here we will use the *SAT* backend, see [MN10]). Once everything is correct, press the OK button.



Depending on your computer, you have to wait for some seconds before the solution is printed :



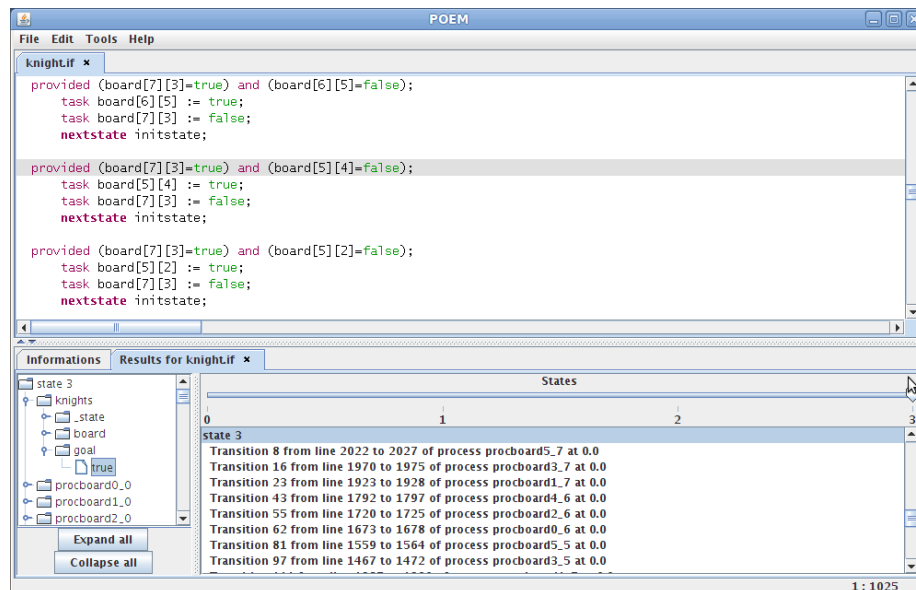
On the left part of the solution, a tree displays the value of the different variables for all the threads. On the right part you can see the different states of the solution, and the different transitions that was executed. Clicking on a transition highlight the corresponding source-code. Let's examine a transition:

Transition 172 from line 1019 to 1023 of process `procboard7_3` at 0.0

The different elements are:

- The number of the transition (e.g `Transition 172`)
- The lines in the source-code of the transtion (e.g `line 1019 to 1023`)
- The process (the thread) of the transition (e.g `procboard7_3`)
- The timestamp, i.e the moment when the transition was executed (e.g `at 0.0`). It has no meaning if there is no clock in the model (untimed model): this is the case for the knights problem.

One should notice that several transitions were executed from state 0 to state 1. These transitions are executed in parallel (see [MN10]). The last state (here state 3) is the state where the query is verified.



References

- [MN10] Janusz Malinowski and Peter Niebert. SAT based bounded model checking with partial order semantics for timed automata. In *TACAS 2010 – 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 6015 of *Lecture Notes in Computer Science*, pages 405–419. Springer, 2010.