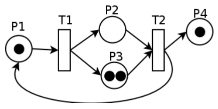
ADT : proofs and rewriting



October 22nd 2015

David Lawrence Edmundo Lopez Alexis Marechal

Final date : November 4th 2015, at 23:55

Files to include : Report in the `pdf` format. No other file is needed!

The homeworks are *personal*, if two solutions have flagrant similarities, they will both receive a grade of 0.

The final date and time are *strict*, any delay (even by one minute) will be penalized following the instructions given at the beginning of the course. This date is expressed in Geneva local time.

Be careful about the presentation of the homework. The report will always receive a grade. Minor problems (such as the file formats, name of the student in every file) can have a negative impact on the grade of the homework.

This TP will cover two important aspects in the domain of ADTs: proofs (both deductive and inductive) and rewriting systems. Please note that this TP is *not* meant to be solved in AlPiNA, as both subjects are out of the scope of the tool. All your proofs must be presented *in detail* in your report. The second exercise is purely a theoretical question. As for the third exercise, you are invited to use the tool *stratagem* to improve your reasoning and your answers to the questions.

Quick introduction to *stratagem*

For this TP, you must install *stratagem*. This tool can be download at `http://sourceforge.net/projects/stratagem-mc/` and install instructions are to be found at `https://github.com/mundacho/stratagem` under the **requirements** section. Normally, you should just need to have Java 1.7 or later installed.

This tool accepts various file formats as input but we'll focus on using a transition system.

Definition : A transition system is a triple $T =< S, \rightarrow, s^0 >$ where :

— S is a set of states ;

— $\rightarrow \subseteq S \times S$ is a transition relation ;

— $s^0 \in S$ is the initial state.

In our case, the states represent terms and the arcs given in the transition relation are rewriting rules / strategies (defined under the keyword *Transitions*).

The file *halloween.ts* is structured as follow :

— The keyword *TransitionSystem* ;
— The string *ADT + [yourADT name]* ;
— The keyword *Sorts* followed by a list of sorts ;
— The keyword *Generators* followed by a list of generators ;
— The keyword *Variables* followed by a list of variables ;
— An initial state named *initialState* ;
— The keyword *Strategies* followed by a list of strategies ;
— The keyword *Transitions* followed by a list of transitions.

As for the tool execution, inside the folder *stratagem-0.4.1/bin*, there is an executable for both Windows, named *stratagem.bat*, and UNIX systems, named *stratagem*. To execute the tool, use a terminal and write one of the following command lines :

stratagem transition-system halloween.ts
or
stratagem.bat transition-system halloween.ts

Executing the tool with the provided file should return the following result :

```
Successfully created initial state
Successfully created state space rewriter
Starting calculation of state space
Elapsed time: 0.252365[seconds]
Total memory used: 439[MB]
Total cache hits: 251
Total rewrites: 756
Cache hits to rewrites ratio: 33%
State space size:
7
```

In some questions of the exercise 3, the information *State space size* should be used to explain your answers. The state space can be seen more or less as a reachability graph.

As a final remark, *stratagem* provides a deterministic version of the *One* strategy seen in the course. This strategy is defined as follow :

$$One(s,i)[f(t_1,...,t_i,...,t_n)] = f(t_1,...,t'_i,...,t_n), 1 \leq i \leq n \Leftarrow (strat)[t_i] = t'_i$$
$$One(s,i)[f(t_1,...,t_n)] = fail, 1 \leq i \leq n \Leftarrow (strat)[t_i] = fail$$
$$One(s,i)[cst] = fail$$

Exercise 1 : Proof

Make a detailed and precise proof of the following theorem by using the ADTs in listings 1 and 2. You will need a non trivial combination of equational proofs and inductive proofs. Do not forget to indicate all the axioms that you use, including the symmetry and transitivity for the equational proofs. Please note that you will have to adapt the theory slightly in order to cope with conditional equations.

```
s($x) divide s($x) = s(zero)
```

Exercise 2 : Rewriting systems and Stratagem

1. When defining a rewriting system using an algebraic specification, what would happen if you chose the wrong orientation for your rewriting rules with the axioms ? Use the given specification (1 and 2) in order to justify your answer.

Listing 1 – Booleans ADT

```
ADT Bool;
    Sorts
       bool;
    Generators
       true : bool;
       false : bool;
    Operations
       not : bool → bool;
       and : bool, bool → bool;
       or : bool, bool → bool;
       implies : bool, bool → bool;
    Axioms
        //not
        false = not(true);
        true = not(false);

        //and
        $x and true = $x;
        false = $x and false;

        //implies
        $x implies ($y and $z) = ($y and $z) or not($x);
        false implies $x = true;
        $x = true implies ($x and true);
    Variables
        x : bool;
        y : bool;
        z : bool;
```

Listing 2 – Naturals ADT

```
import "Bool.adt"

ADT Nat;
   Sorts
      nat;
   Generators
      zero : nat;
      s : nat → nat;
   Operations
       plus : nat, nat → nat;
       minus : nat, nat → nat;
       ge : nat, nat → bool;
       times : nat, nat → nat;
       divide : nat, nat → nat;
   Axioms
       //plus
       $x plus zero = $x;
       $x plus s($y) = s($x plus $y);

       //minus
       $x minus zero = $x;
       s($x) minus s($y) = $x minus $y;

       //ge: greater or equal
       $x ge zero = true;
       zero ge s($x) = false;
       s($x) ge s($y) = $x ge $y;

       //times
       $x times zero = zero;
       $x times s($y) = $x plus ($x times $y);

       //divide
       //Please excuse the heresy of defining x/0 = 0
       if ge($x, $y) = true then
           $x divide $y = s(($x minus $y) divide $y);
       if ge($x, $y) = false then
           $x divide $y = zero;
   Variables
       x : nat;
       y : nat;
```

Exercise 3 : The terrifying journey of a little girl

We'll use the tool *stratagem* and the *halloween.ts* file provided on moodle for this exercise. Moreover, we'll use the following textual specification called *The terrifying journey of a little girl* to understand the modelled problem.

A little girl just woke up from a horrible nightmare. In this nightmare, she was hunted by a group of ghosts. She is very afraid and wants to get to her parents' bedroom to be comforted and to be protected from the ghosts. However, her parents' bedroom is at the very end of a dark corridor and at the opposite of her room. Moreover, the switch to light up this corridor is next to her parents' bedroom door. Therefore, she must cross the entire corridor in the darkness to reach her parents' bedroom. Since she is afraid, she takes one step after the other. If a ghost seems to appear in front of her, she just freezes due to the fear and will not move. Finally, when she reaches the end of the corridor, she is able to switch on the light and enter her parents' bedroom.

As a summary, the behavior of the little girl is as follow :
- The first step of the little girl is to get out of her bedroom ;
- If there is no ghost in the next section of the corridor, she will walk to it (figure 2) ;
- If there is a ghost in the next section of the corridor, she will freeze and close her eyes. Therefore, she will not walk to the next section of the corridor (figure 3) ;
- When she reaches the light switch at the end of the corridor, she will light up the whole corridor (figure 4).

Ghosts have the follow behavior :
- A ghost will scare the little girl if she wants to come to his section of the corridor and then disappear forever (figure 3).

Initially, the little girl position is obviously defined at first : the furthest from the light switch, in her room. The ghosts are initially positioned in the corridor and will scare the little girl only if she tries to move to their section of the corridor.

The following *initialState* can be alternatively seen as the picture depicted on the figure 1 :

```
initialState = door_open(afraid_girl, dark_corridor(
    empty_space, dark_corridor(
        empty_space, dark_corridor(
            empty_space, dark_corridor(
                empty_space, dark_corridor(
                    empty_space, light_switch))))))
```

Now that the problem has been clearly defined, provide detailed answers for each questions :

1. Why do we need to apply the strategy *One* before applying the strategy *Try_to_reach_light_switch_with_fear* in the transition *conti-*

nue_the_terrifying_journey_with_fear ? What would happen if we didn't use the strategy *One* in the latter case ?

2. Explain the goal and the semantic of the strategy *ForEachRoom(V)* out of the problem context.
3. For the given *initialState*, when we construct the problem state space, we end up with 7 states. Explain why.
4. What is the effect on the state space if we had a ghost in the third empty space of the corridor as follow ?

```
initialState = door_open(afraid_girl, dark_corridor(
    empty_space, dark_corridor(
        empty_space, dark_corridor(
            ghost, dark_corridor(
                empty_space, dark_corridor(
                    empty_space, light_switch))))))
```

5. What is the effect on the state space if we had a ghost in the first empty space has follow ?

```
initialState = door_open(afraid_girl, dark_corridor(
    ghost, dark_corridor(
        empty_space, dark_corridor(
            empty_space, dark_corridor(
                empty_space, dark_corridor(
                    empty_space, light_switch))))))
```

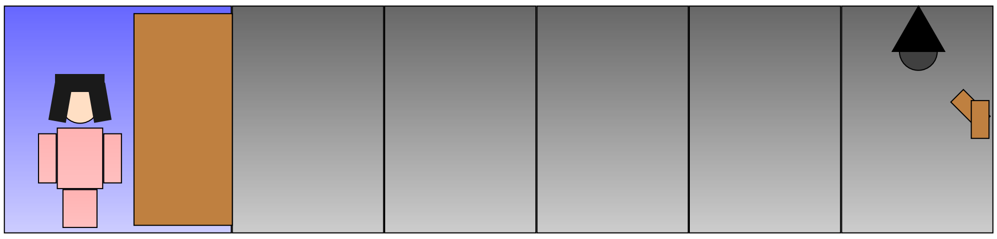


FIGURE 1 – Possible initial state (with no ghosts)

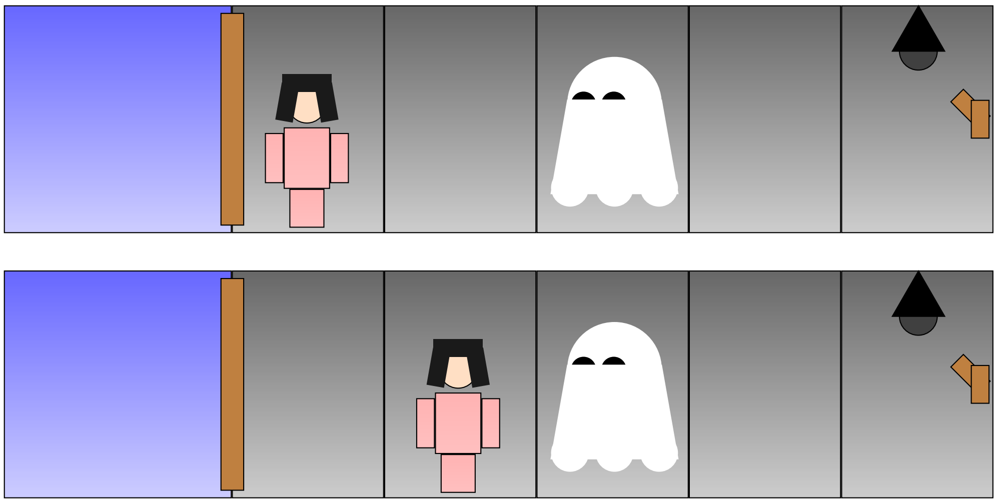


FIGURE 2 – Take a step strategy : before and after

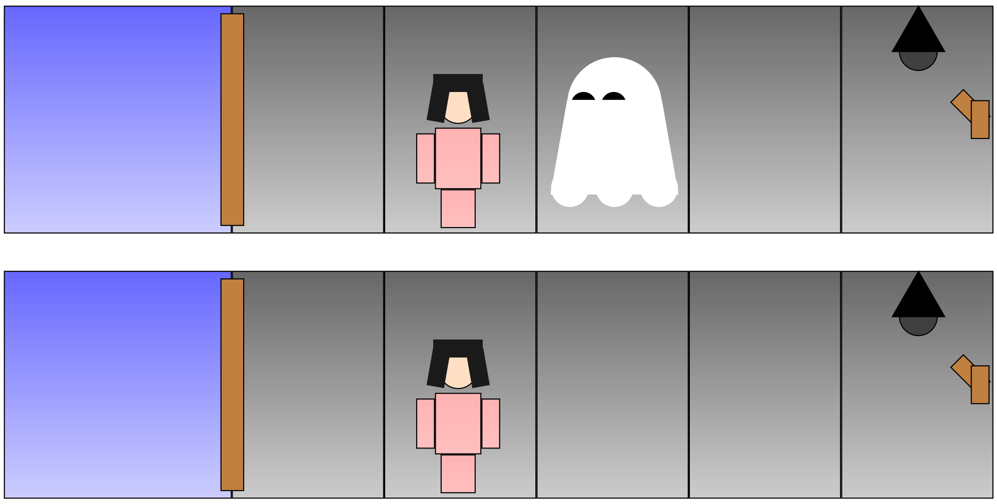


FIGURE 3 – Freeze strategy : before and after

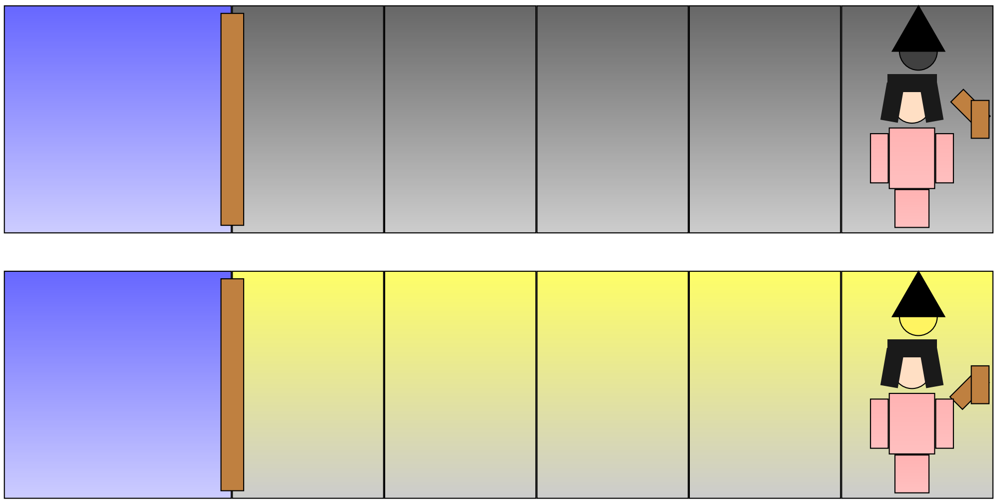


FIGURE 4 – Ending the terrifying journey strategy : before and after