VSTTE 2010 software verification competition

Problem 4: N-Queens

I. Description and analysis

Problem:

Write a program to place N queens on an N*N chess board so that no queen can capture another one with a legal move.

The algorithm returns a placement if there is a solution and an empty board otherwise. You can represent the empty board with a flag or a null pointer.

A placement is given by a board which is an N-element array where the i'th element is j, when the queen is placed in the j'th row for the i'th column.

Properties:

The post-condition should establish that when the algorithm returns a placement, it is legal, and if it returns an empty board, there is no solution.

Analysis and adaptation of the problem:

We want a program that:

- gives a valid placement if any;
- proves there are no placement possible if not.

First of all, recursive algorithms are not possible in B since it is impossible to prove the termination of the algorithm. An equivalent iterative of this algorithm is possible ad is made of two imbricated loops.

Proving that all the possibilities were tested when there are no solutions is not easy in B. Indeed, finding invariants for the loops will be complex.

Do you know what event-B is?

Find it out there: http://www.event-b.org/

Why will we chose event-B instead of B?

In our case event-b is mostly used for the proof that if there are no solutions then we have tested all the possibilities.

Systems in event-B haven't an implementation, but we have the possibility to animate those using tools like ProB.

The problem is solved using the Rodin Platform, but it is possible to do it using the Atelier B: when creating a new project, select 'System modeling' instead of 'Software development'.

Rodin and the Atelier B are equivalent; archives of both will be given.

You may find useful to install other plugins for the Rodin platform.
Go to *Help > Install new software…* and select software to install:

- Camille text editors
- Atelier B prover
- ProB

Click on the *Select All* button, and finish the installation clicking on the *Next* buttons and accepting the terms and conditions.

Camille is a text editor for the Rodin plug-in. For the anecdote, the name comes from:
http://en.wikipedia.org/wiki/Camille_Claudel

Atelier B provers are used for the proof of POs.

ProB is needed to animate the system.

II. System modeling

Two constructions are possible for the system:

- 'Extracting' events from the iterative algorithm (like 'Add_a_queen', 'Remove_queen', 'move_queen' …): this way proving that all the possibilities where tested when there are no solutions is far more complex than in B;
- Building a simple system: This is the chosen option, it has benefits and disadvantages. Those are explained in the following section.

A. Building the system

The system is very easy: we define two events: solution and no_solutions.
The guards are easy:

- The existence of a solution can be defined like:

```
P :  #b.(b: 1..N -->1..N &
            !(ii,jj).(ii: dom(b) & jj: dom(b) & jj<ii
                    => (not(b(ii)=b(jj)) &
                        not(b(ii)-b(jj)=ii-jj) &
                        not(b(jj)-b(ii)=ii-jj))
            )
   )
```

"There is a board such that no queen can capture another one".
Note that this expression P can be simplified. Note also that the system just built is not optimized since all the guards are evaluated when animating: the job is done for every event using P.

If there is a solution we also have to keep a placement: the only way to do it is to use the ANY substitution.

The system created under Rodin looks like:

```
machine BackTrack sees Constants

variables board

invariants
  @inv1 board ∈ 1‥NN → 1‥NN

events
  event INITIALISATION extends INITIALISATION
    then
      @act1 board ≔ (1‥NN) × {1}
  end

  event solution
    any b
    where
      @grd3 P
    then
      @act1 board ≔ b
  end

  event no_solutions
    where
      @grd1 not(P)
  end
end
```

The variable N in rodin his defined in a context machine.
The *b* variable in the *solution* event is the same as in P: The # quantification symbol has to be removed.

This system sketch has to be simplified.

B. Simplifying the system

We have now to simplify two things:

1: Add an 'initialisation' event that computes P into a Boolean;

2: Reduce the expression of P: Since ProB (for the animation) is in Prolog, the more the expression is reduced the more efficient it is.

1) Initialisation event

We add a new boolean variable *solution* that will indicates if there is a solution for N Queens on the NxN board.
First create an event 'initialisation' that is made of an action:

sol := bool(P).

The problem is that this event can be taken any time since it has no guards.
We only want this event to be available at the beginning of the animation: we can add another boolean variable (let call it '*init*') that is initialized at FALSE. Add the guard 'init = FALSE' in the event initialization and don't forget also to add an action: 'init := TRUE'.

Add the guard '*init* = TRUE' to the two events *solution* and *no_solutions* so there are taken after the initialization is done. Modify also the previous guards P and not(P) because they has the same value as the *solution* boolean: P becomes 'solution = TRUE' and not(P) becomes 'solution = FALSE'.

2) Reducing P

We have:

```
P  =  #b.(b: 1..N -->1..N &
            !(ii,jj).(ii: dom(b) & jj: dom(b) & jj<ii
                        => (not(b(ii)=b(jj)) &
                             not(b(ii)-b(jj)=ii-jj) &
                             not(b(jj)-b(ii)=ii-jj))
            )
   )
```

We can divide the

!(ii,jj).(ii: dom(b) & jj: dom(b) & jj<ii =>
(not(b(ii)=b(jj)) & not(b(ii)-b(jj)=ii-jj) &not(b(jj)-b(ii)=ii-jj)))

Let's define:

①: !(ii,jj).(ii: dom(b) & jj: dom(b) & jj<ii =>
(not(b(ii)=b(jj))))

②: !(ii,jj).(ii: dom(b) & jj: dom(b) & jj<ii =>
(not(b(ii)-b(jj)=ii-jj) &not(b(jj)-b(ii)=ii-jj)))

So we obtain P = #b.(b: 1..N -->1..N & ① & ②).
But 'b: 1..N -->1..N & ①' is the definition of the injectivity ! We now rewrite P:

P = #b.(b: 1..N >->1..N & ②)

Note that "ii: dom(b) & jj: dom(b) & jj<ii" is equivalent to "ii: bom(b) & jj: 1..ii-1".

The new system looks like:

```
machine BackTrack sees Constants

variables board sol init

invariants
  @inv1 board ∈ 1‥NN → 1‥NN
  @inv2 sol ∈ BOOL
  @inv3 init ∈ BOOL

events
  event INITIALISATION extends INITIALISATION
    then
      @act1 board ≔ (1‥NN) × {1}
      @act2 sol ≔ FALSE
      @act3 init ≔ FALSE
  end

  event solution
    any b
    where
      @grd1 sol = TRUE
      @grd2 init = TRUE
      @grd3 b∈ 1‥NN ⤖ 1‥NN ∧
                        (∀ii,jj·(ii∈ 1‥NN ∧ jj∈ 1‥ii−1 ⇒
                               (¬(b(jj)−b(ii)=ii−jj) ∧ ¬(b(ii)−b(jj)=ii−jj))))
    then
      @act1 board ≔ b
  end

  event no_solutions
    where
      @grd1 sol = FALSE
      @grd2 init = TRUE
  end

  event initialSol
    where
      @grd1 init = FALSE
    then
      @act1 sol ≔ bool(∃b·(b∈ 1‥NN ⤖ 1‥NN ∧
                        (∀ii,jj·(ii∈ 1‥NN ∧ jj∈ 1‥ii−1 ⇒
                               (¬(b(jj)−b(ii)=ii−jj) ∧ ¬(b(ii)−b(jj)=ii−jj))))))
      @act2 init ≔ TRUE
  end
end
```

C. Proving the system

4 POs are generated, and all the 4 are automatically discharged.

Those POs do not concern the deadlock freeness of the system.

D. Advantages and disadvantages

The advantages of such a system are:

- Easy to build (regarding the property to prove);
- Easy to prove (all POs discharged automatically).

The disadvantages are:

- Algorithm use not known (but it surely is brute force);
- Even with this architecture, if there is a solution, the same job is done three times (at the initialization: seek if there is a solution, when the animation computes the guard of the event *solution*, and when the event is taken)

III. System animation

A. Configure ProB

The configurations for ProB are found in *Window > Preferences > ProB*.

Change the value of 'Time out for computing enable transitions (in ms)'. You can set it to long for a few minutes according to the number of queens.

You can also use the symmetry mode. Details on the parameters are available here:
http://www.stups.uni-duesseldorf.de/ProB/index.php5/Symmetry_Reduction

If only one solution is needed, set the 'Max Number of Enablings per Operation Computed' to one. If you want all the solution possible for a board, you can set to 10000 or over (The more there are the longer it takes).

Select also the checkbox 'Use more aggressive COMPRESSION when storing states'.

B. Animate

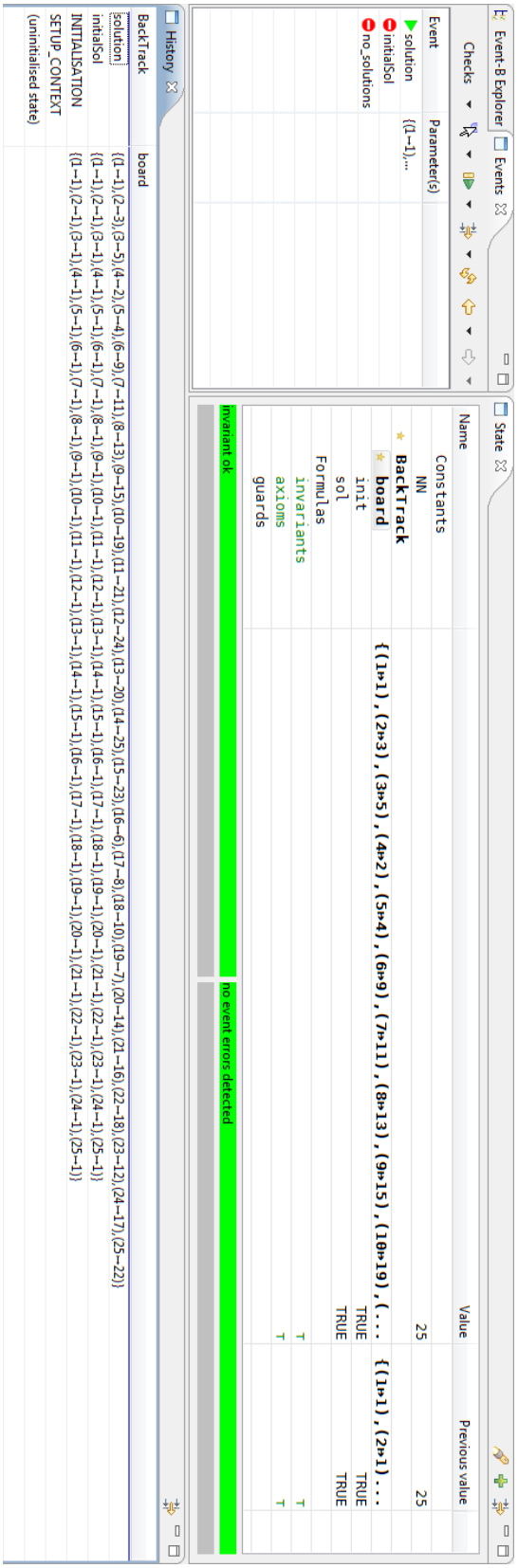
Right click on the event-B component and select *Start Animate/Model Checking*, this will normally open a few tabs: events, state and history.
You can set the constants before animating the system.

You can now click on the available events, and see the variables' state change according the events. Seeking for a solution when there is more than 25 queens takes time, but this is the price to pay for having such a property to prove (that there are absolutely NO solutions if the board returned is the initial board).

Execution times are exponential: adding a queen will take a lot more time (compared to the current execution time) to compute.

Event-B Explorer
Events
Checks
Event
Parameter(s)
solution
{(1↦1),...
initialSol
no_solutions
History
BackTrack
solution
initialSol
INITIALISATION
SETUP_CONTEXT
(uninitialised state)
State
Name
Value
Previous value
Constants
NN
8
8
BackTrack
board
{(1↦1),(2↦5)...
{(1↦1),(2↦1)...
init
TRUE
TRUE
sol
TRUE
TRUE
Formulas
invariants
T
T
axioms
T
T
guards
invariant ok
no event errors detected

Event-B Explorer
Events
Checks
Event
Parameter(s)
solution
initialSol
no_solutions
History
BackTrack
no_solutions
initialSol
INITIALISATION
SETUP_CONTEXT
(uninitialised state)
State
Name
Value
Previous value
Constants
NN
3
3
BackTrack
board
{(1↦1),(2↦1),(3↦1)}
{(1↦1),(2↦1),(...
init
TRUE
TRUE
sol
FALSE
FALSE
Formulas
invariants
T
T
axioms
T
T
guards
invariant ok
no event errors detected


Event-B Explorer
Events
Checks
Event
Parameter(s)
solution
{{1–1},...
initialSol
no_solutions
State
Name
Value
Previous value
Constants
NN
25
25
BackTrack
board
{(1↦1),(2↦3),(3↦5),(4↦2),(5↦4),(6↦9),(7↦11),(8↦13),(9↦15),(10↦19),(...
{(1↦1),(2↦1)...
init
TRUE
TRUE
sol
TRUE
TRUE
Formulas
invariants
T
T
axioms
T
T
guards
invariant ok
no event errors detected
History
BackTrack
board
solution
{(1–1),(2–3),(3–5),(4–2),(5–4),(6–9),(7–11),(8–13),(9–15),(10–19),(11–21),(12–24),(13–20),(14–25),(15–23),(16–6),(17–8),(18–10),(19–7),(20–14),(21–16),(22–18),(23–12),(24–17),(25–22)}
initialSol
{(1–1),(2–1),(3–1),(4–1),(5–1),(6–1),(7–1),(8–1),(9–1),(10–1),(11–1),(12–1),(13–1),(14–1),(15–1),(16–1),(17–1),(18–1),(19–1),(20–1),(21–1),(22–1),(23–1),(24–1),(25–1)}
INITIALISATION
{(1–1),(2–1),(3–1),(4–1),(5–1),(6–1),(7–1),(8–1),(9–1),(10–1),(11–1),(12–1),(13–1),(14–1),(15–1),(16–1),(17–1),(18–1),(19–1),(20–1),(21–1),(22–1),(23–1),(24–1),(25–1)}
SETUP_CONTEXT
(uninitialised state)