INTEGRATION OF AUTOMATIC THEOREM PROVERS IN EVENT-B PATTERNS

Eman Elsayed[1], Gaber El-Sharawy[2] and Enas El-Sharawy[3]

[123]Faculty of Science(girls), Al-Azhar University, Cairo, Egypt

emankaram10@azhar.ed.eg , gelsharawy@gmail.com, dr_enas_idm@hotmail.com

ABSTRACT

Event-B is a formal method for the system level modeling and analysis of dependable applications. It is supported by an open and extendable Eclipse-based tool set called Rodin. In this paper we proposed using Automatic theorem provers known as SMT-solvers with event-B pattern. The benefits of that are to reduce the proving effort, to reuse a model and to increase the degree of automation. The proposed approach has been applied successfully on two different case studies.

KEYWORDS

Formal methods, Event-B, Design patterns, SMT-Solver, Refinement.

1. INTRODUCTION

Nowadays, there are many formal methods used in various domains for constructing models of complex system together with a several advanced theories and tools. But more experiments in this area are still needed to be carried out to provide significant evidence for convincing and encouraging other users to benefit from those theories and tools, and make formal methods more accessible to software industries. However, there is no systematic approach for refinement model with formal methods so we design pattern. Since refinement model is monotonic, the final resulting model shall be a refinement of the original model. Practically, but the last refinement model of the pattern's refinement-chain is incorporated in the development. This feature allows us to reuse formal models more flexibly.

Also when used formal methods as event-b, produces large quantities of proof obligations (Pos). For each such PO can be valid, but cannot be automatically proved because the proof system lacks some axioms, as the specification logic is incomplete. In that case, it is sometimes possible to patch the prover with additional rules so that it finds a proof for the verification condition.

The goal of the work presented in this paper is to study the possibility of enhancing event-b patterns. We want to bring the Satisfiability Modulo Theory (SMT) into formal methods and in particular SMT-solvers tools into event-B patterns. It considers the POs generated in the Rodin Platform and using SMT-solvers and Pattern design tools. This approach should also be readily applied in other development environments.

For our purposes we apply the proposed approach on two case studies of a sequential algorithms first binary search algorithm. The second case study is finding the minimum value of an array of natural numbers. We design pattern and prove the correctness of binary search algorithm automatically discharges a part of the proof obligations and also provides the interactive prover to discharge the remaining proofs. The rest of the paper is organized as follows. The next section presents the main concepts then section three introduces the literature review. The proposal approach is described in section four. The approach is applied on two case studies in section five and the result analyzed in section six. Conclusions and future work are discussed in section seven. Finally, list the references.

2. MAIN CONCEPTS:

This section presents the background material for the paper. We start by overviewing the Event-B formalism. Then we describe the RODIN platform tool.

2.1. Event-B and Rodin

The B Method in reference [3] is a formal approach for the specification and rigorous development of highly dependable software. The method has been successfully used in the development of several complex real-life applications [5]. Event-B[10] is a formal framework derived from the B Method to model and reason about parallel, distributed and reactive systems. Event-B has the associated RODIN platform [4,5,6], which provides automated tool support for modeling and verification by theorem proving. Then Event-B is an extension of the B-method for specifying and reasoning about complex systems including concurrent and reactive systems. An Event-B model is described in terms of contexts and machines as shown in Figure 1.

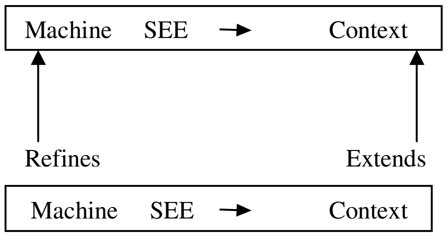


Fig.1. Machine and context relationship

Where, contexts contain the static parts of a model [1]. Each context may consist of carrier sets and constants as well as axioms which are used to describe the properties of those sets and constants. Contexts may contain theorems for which it must be proved that they follow from the preceding axioms and theorems. Moreover, contexts can be extended by other contexts and seen by more than one machine.

Machines contain the dynamic parts of an Event-B model [1]. This part is used to provide behavioral properties of the model. A machine is made of a state, which is defined by means of variables, invariants, events and theorems. A machine contains a number of atomic events which show the way that the machine may evolve. Each event is composed of three elements: an event name, guard(s) and action(s). The guard is the necessary condition for the event. The actions determine the way in which the state variables are going to evolve when performing the event [1]. In addition, machines can be refined by other machines, but each machine can refine at most one machine.

Machine refinement provides a means to introduce more details about the dynamic properties of a model [2]. For more on the well-known theory of refinement, we refer to the Action System formalism that has inspired the development of Event-B [5]. We present some important proof obligations for machine refinement. As mentioned before, the user of Event-B is not presented with a behavioral model but only with proof obligations. The proof obligations describe the semantics of Event-B models.

The Rodin platform is an open and extensible tool for Event-B specification and verification [3,18]. It contains a database of modeling elements used for constructing system models such as variables, invariants and events. It is accompanied by various useful plug-ins such as a proof-obligation generator, provers, model-checkers, UML transformers, etc .

2.2 Design Pattern

The intention of Design Patterns in Event-B is to have a methodological approach to reuse former developments (referred to as patterns) in a new development. The patterns approach is a promising avenue to let inexperienced designers build conceptual models and as a tool for building domain models [21]. The proofs of the pattern can be reused too. It was shown that for the special case of a model that does not see any context and its events do not have any parameters, the generation of a refinement of the problem at hand is correct by construction and no proof obligation needs to be generated again. The correctness of the construction relies on a correct matching of the pattern with the problem at hand. We summarize the incorporating patterns into Event-B developments:

First of all, in our notion, a pattern is just a development in Event-B including specification p0 and a refinement P1. In P0 we introduce a variable r_pat with event final_pat. P1 will be like the second refinement that in the section 5.1.3 but by another naming for each component in Event-B. As a first step, all the variables of the pattern specification have to be matched with variables of the problem. Furthermore all the events of the pattern have to be matched with a event of the problem. The chosen matching is valid for the construction if the following checks turn out to be true. Note that this is only formally proved for models that do not see any context or having events with parameters. The proof of the correctness of the construction with general models is pending.

- Check that the guards of each event of the pattern specification are syntactically the same as the guards of the corresponding event in the problem.
- Check that the actions of each event of the pattern specification are syntactically the same as the actions of the corresponding event in the problem.
- Check that no event in the problem that is not matched alters a matched variable.

Once all checks are done, the refinement of the problem is generated by merging the pattern refinement with the problem.

2.3 Satisfiability Modulo Theory (SMT)

The Satisfiability Modulo Theory (SMT) problem is a decision problem to determine if a given logic formula is satisfiable with respect to a combination of theories expressed in first-order logic. Theories of interest for the work described in this paper include uninterpreted functions with equality and integer arithmetics. Since the validity of a proof obligation can be decided by checking the unsatisfiability of its negation, SMT-solvers are natural candidates for discharging the verification conditions generated in the application of Event-B methods. SMT-solvers can for example handle a formula like

$$x \leq y \land y \leq x + f(x) \land P(h(x) - h(y)) \land \neg P(0) \land f(x) = 0$$

Which contains linear arithmetic on real's (0,+,–, ≤), and un-interpreted symbols (P,h,f) . SMT-solvers use decision procedures for the disjoint languages (for instance, congruence closure for un-interpreted symbols, and simplex for linear arithmetic) and combine them to build a decision procedure for the union of the languages. There are a number of available SMT-solvers as in references [8 ,9,15,12]. They differ mostly by the theories they handle, their efficiency (which may vary depending on the theories they handle). The ability to handle quantified formulas as well as to construct certificates (proofs) of their results.

3. LITERATURE REVIEW

Event-B is a language for the formal development of reactive systems. At present the RODIN toolkit [5] for Event-B is used for modeling requirements, specifying refinements and verification. The design pattern is not a new idea and is not restricted to the field of computer science. Design patterns were introduced by Christopher Alexander in the field of architecture. In 1977 he spoke of patterns as, "each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice"[AIS+77]. Event-b pattern[17] approach has been applied to formalize communication protocols from SAP. The examples are Buyer/Seller B2B as described in [20] and Ordering/SupplyChain A2A Communications as described in [14, Section 5.3.3].

The proof statistics compare with the developments without patterns and with patterns for the two case studies. More importantly, this approach saves on average of the two case studies 33% of the manual proofs). When applying SMT-solvers on the European project Deploy by the developments made publicly available Web site (http://www.deploy-project.eu/) , all the proof obligations generated by the developments was 2359 proof obligations, and it terminated successfully in 54% of the cases. When applying the SMT-Solver approach, the ratio of discharged proof obligations jumps to 63%. It is important to note that all the runs of the SMT-solver take a negligible amount of time. The remaining proof obligations shall be communicated to the community of researchers on SMT-solving techniques in the hope that use them as a direction for future extensions and improvements [13].

4. THE PROPOSAL APPROACH

The proposed approach is classified into two main phases:

First phase, design the event-B pattern to the final refinement model. That is for guarantee the reusability in any similar models.

Second phase, apply Automatic theorem provers known as SMT-solvers with event-B pattern. That is for guarantee reducing the proving effort and increasing the degree of automation.

The proposed approach has been applied successfully on two case studies. First is a binary search model and the second is Minimum of array model. Each model is created in Rodin platform application once completely and by using design pattern approach.

5. CASE STUDIES

5.1 Overview of a Binary Search:

A binary search or half-interval search algorithm finds the position of a specified value (the input "key") within a sorted array. The algorithm compares the input key value with the key value of the middle element of the array. If the keys match then a matching element has been found so its index, or position, is returned. Otherwise, if the sought key is fewer than the middle element's key, then the algorithm repeats its action on the sub-array to the left of the middle element or, if the input key is greater, on the sub-array to the right. If the remaining array to be searched is reduced to zero, then the key cannot be found in the array and a special "Not found" indication is returned. A binary search halves the number of items to check with iteration so locating an item (or determining its absence) takes logarithmic time. A binary search is a dichotomist divide and conquer search algorithm.

5.1.1 Initial Model

In this abstraction, we begin with an abstract model of binary search by given (Pre-condition) a natural number n: n ∈ ℕ, n is positive: 0<n, a sorted array f of n elements built on a set ℕ: f ∈ 1..n! ℕ, a value v known to be in the array: v ∈ ran(f). We are looking for (Post-condition) an index r in the domain of the array: r ∈ dom(f) such that f(r) = v. Two events are defined called final and progress.

```
Constants: n, f, v
```

```
Inv1:   r :∈ ℕ
```

```
Axioms
axm1:   f ∈ 1··n → N
axm2:   ∀i,j· i∈1··n ∧ j∈1··n ∧ i≤j
                      f(i)≤f(j)
axm3:   v ∈ ran(f)
```

```
      Final
WHEN
        grd2 : r ∈ 1··n
        grd1 :  f(r) = v
THEN
```

5.1.2 First Refinement

First refinement consist in introducing two new variables p and q. Variables p and q are supposed to be two indices in the array f (inv1_1 and inv2). The variable r is within the interval p .. q (inv3). Moreover, the value v is supposed to be a member of the set denoting the image of the interval p .. q under f: that is, f[p .. q] (inv4). Here is the state of this refinement:

```
Constants:  n,f,v
Variables:  r,p,q
```

```
Inv1-1: p ∈ 1··n
Inv2: q ∈ 1··n
Inv3: r ∈ f [p.. q]
```

Now, we introduce two events called inc and dec which split abstract anticipated event progress. These events are convergent (see variant1 above). They are increment or decrement p or q when f(r) is smaller or greater than v. They also move r non deterministically within the new interval p .. q: The total events of first refinement

```
Init
p ≔ 1
q ≔ n
```

```
Increment
When
F ( r )< v
then
p ≔ r+1
```

```
decrement
When
v < F ( r )
then
q ≔ r-1
```

```
final
When
 F ( r ) = v
then
skip
```

5.1.3 Second Refinement

At the previous stage, Increment and decrement were non-deterministic and r was chosen arbitrarily within the interval p .. q. We now remove the non-determinacy in Increment and decrement and r is chosen to be the middle of the interval p .. q.

```
Init
p ≔ 1
q ≔ 1
r ≔ (1 + n) /2
```

```
Increment
When
F ( r )< v
then
p ≔ r+1
```

```
decrement
When
v < F ( r )
then
q ≔ r-1
```

```
final
When
 F ( r ) = v
then
skip
```

5.1.4 Pattern Design

First of all, in our notion, a pattern is just a development in Event-B including specification p0 and a refinement p1

5.1.4.1 Development specification P0

Pattern design specification like initial model in the section 5.1.1

```
Constants: n, f, v
```

```
Axioms
axm1:    f ∈ 1 · · n → N
axm2:    ∀i,j· i∈1 · ·n ∧ j∈1 · ·n ∧  i≤j
                        f(i)≤f(j)
```

```
Inti   r : ∈ N
```

```
        Final
WHEN
        grd2 : r ∈ 1 · ·n
        grd1 :  f(r) = v
THEN
        skip
```

5.1.4.2 Pattern refinement P1

Pattern refinement like first and second refinement in the section 5.1.2 and 5.1.3

```
Variables: pat_r , pat_p, pat_q
```

```
inv1-1: pat_p ∈  1· ·n
inv1-2: pat_q ∈  1· ·n
inv1-3: pat_r ∈ pat_p· ·pat_q
inv1-4: pat_v ∈
```

```
            Inti
BEGIN
        act1 :pat_p ≔ 1
        act2 :pat_q ≔ n
```

```
        Pat_Final
WHEN
        grd1 :pat_f(pat_r) = pat_v
THEN
        skip
END
```

```
            Pat_inc
WHEN
        grd1 :pat_f(pat_r)<pat_v
THEN
        act1 :pat_p ≔ pat_r+1
        act2  pat_r ≔
        :     (pat_r+1+pat_q)÷2
END
```

```
            Pat_dec
WHEN
        grd1 :pat_f(pat_r)>pat_v
THEN
        act1 :pat_q ≔ pat_r−1
        act2 pat_r ≔
        :     (pat_p+pat_r−1)÷2
END
```

5.1.4 .3 Appling Pattern

We have implemented our prototype for supporting our approach as a plug-in for the RODIN Platform [16] which is an open source platform based on Eclipse. The plug-in provides a wizard taking users through different steps of applying patterns, namely, matching, syntax checking, renaming and incorporating. . Fig. 2 is a screen-shot of the wizard page for create new event-b pattern. Fig. 3 is a screen-shot of the wizard page for matching step between the problem and the specification. This step includes a dialog for the developers to choose the matching between variables and events. Fig. 4 is a screen-shot of the wizard page for Syntax checking of the matching provided by the user in the previous steps. Fig. 5 is a screen-shot of the wizard page for renaming pattern of variables and events. Fig 6 is a screen-shot of the wizard page for incorporating the refinement of the pattern into the development.

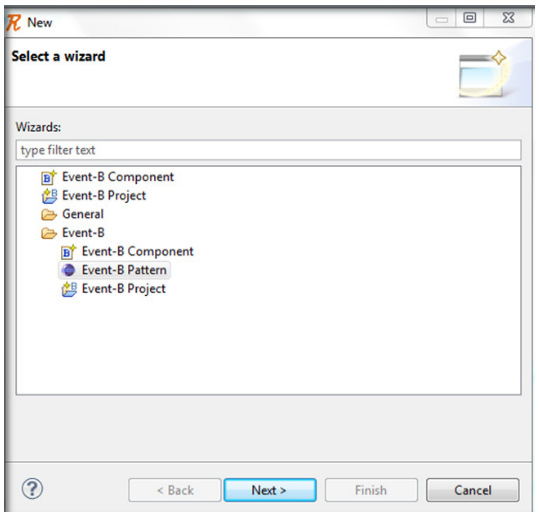
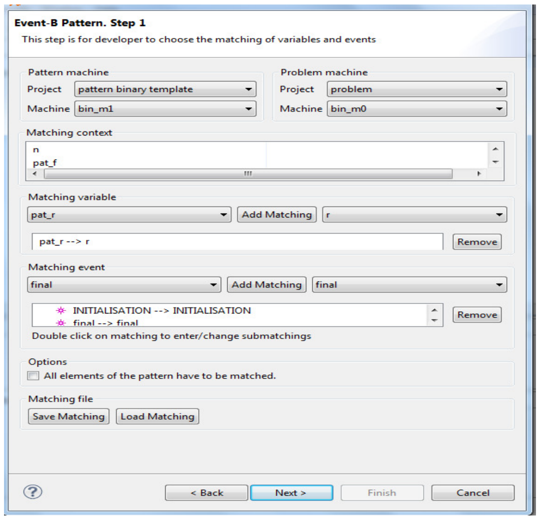


Fig. 2: First step. Begining pattern


Event-B Pattern. Step 1

This step is for developer to choose the matching of variables and events

Pattern machine: Project: pattern binary template; Machine: bin_m1

Problem machine: Project: problem; Machine: bin_m0

Matching context: n, pat_f

Matching variable: pat_r → r

pat_r --> r

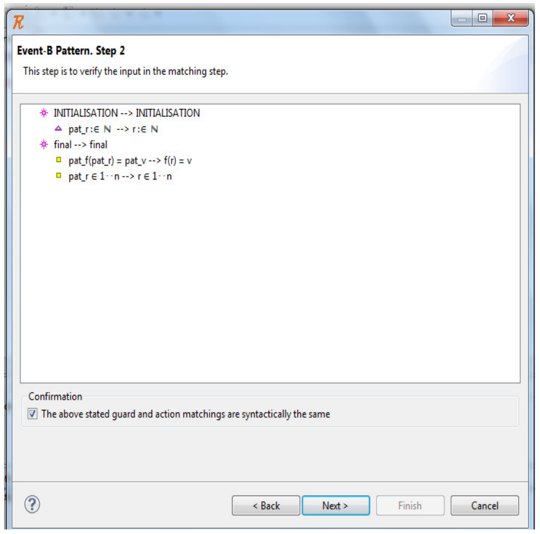
Matching event: final → final

INITIALISATION --> INITIALISATION
final --> final

Double click on matching to enter/change submatchings

Options: All elements of the pattern have to be matched.

Fig. 3 Second step. Matching


Event-B Pattern. Step 2

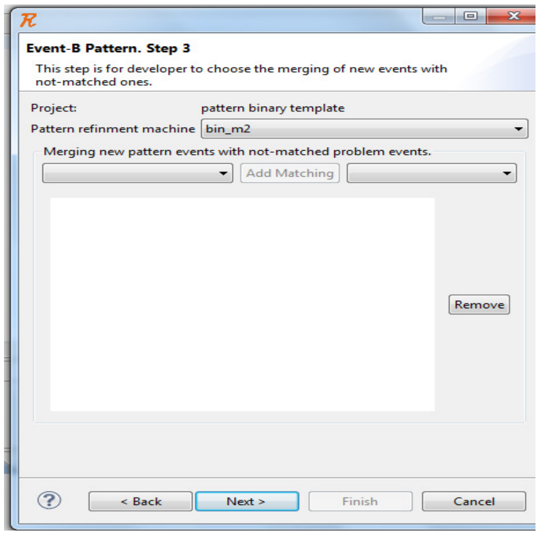
This step is to verify the input in the matching step.

INITIALISATION --> INITIALISATION
pat_r :∈ N --> r :∈ N
final --> final
pat_f(pat_r) = pat_v --> f(r) = v
pat_r ∈ 1‥n --> r ∈ 1‥n

Confirmation

[✓] The above stated guard and action matchings are syntactically the same

Fig4 Third step. Syntax checking


Event-B Pattern. Step 3

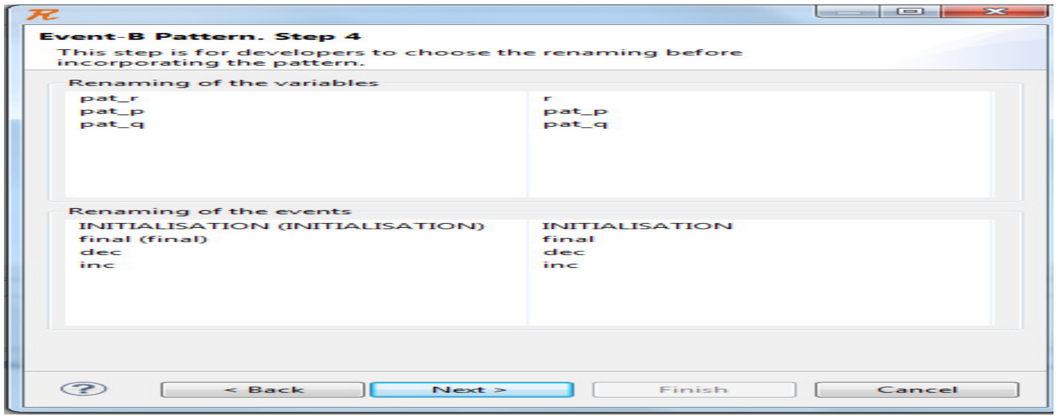
This step is for developer to choose the merging of new events with not-matched ones.

Project: pattern binary template

Pattern refinment machine: bin_m2

Merging new pattern events with not-matched problem events.

Fig. 5 Fourth step. Incorporating



Event-B Pattern. Step 4

This step is for developers to choose the renaming before incorporating the pattern.

Renaming of the variables

pat_r	r
pat_p	pat_p
pat_q	pat_q

Renaming of the events

INITIALISATION (INITIALISATION)	INITIALISATION
final (final)	final
dec	dec
inc	inc

Fig. 6 Fifth step. Renaming

5.1.5 Apply SMT-Solver

The Rodin platform [11] is a development environment for Event-B. It is based on Eclipse [19] and can be extended by developing new plug-ins in the Java programming language. automatic verification of proof obligations using SMT-solvers. The plug-in communicates with the SMT-solvers using files and operating system commands. The configuration of the plug-in includes a choice of SMT-solvers: Alt-Ergo [7], cvc3, veriT [9] and Z3 [12] at this time (see Fig. 7). It is now available to the formal methods community as an exploratory package through Rodin's official source code repository. Currently, the verification with the SMT-solver has to be activated by the user by clicking a button (see Fig. 7, left part). Whenever the verification is successful (see Fig. 7, right part), the status of the proof obligation is updated and the user may move to the next proof obligation.

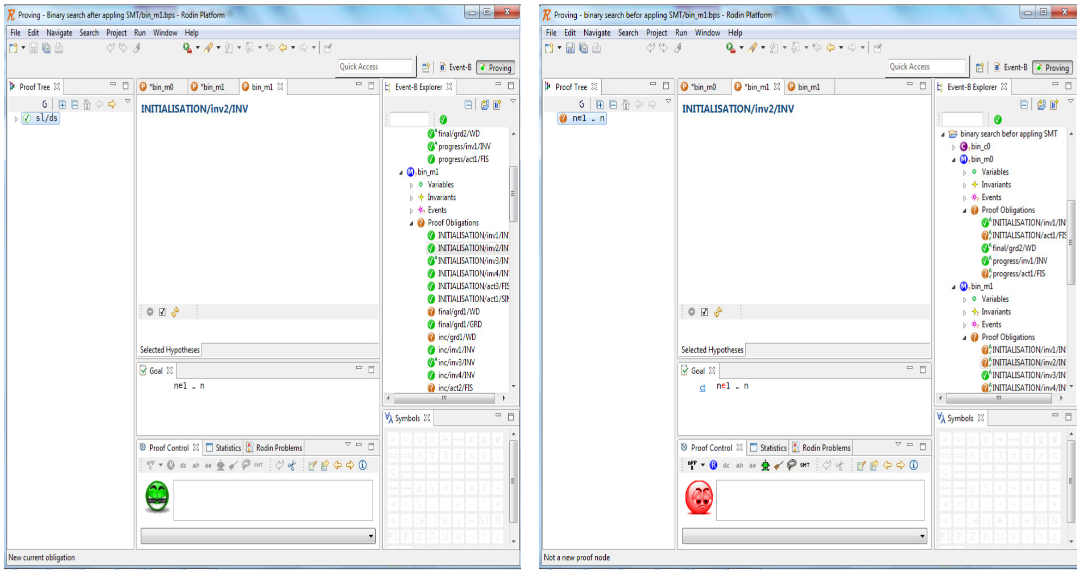


Fig.7. Screen shot of the proof. On the right, before the proof, the button is active and the status is not proved. On the left, after a successful proof, the button has been disactivated and the status is ''proved''.

5.2 Overview of Minimum Model

Minimum model consists in looking for the minimum of the range of a non-empty array of natural numbers.

5.2.1 Initial Model

Let *n* and *f* be two constants, and *m* a variable. Here is our initial model:

Constants: n,f
Variables: m

Inv1: m ∈N

axm1 : $0 < n$
axm2 : $f \in 1 \cdot\cdot n \rightarrow \mathbb{N}$
thm1 : $ran(f) \neq \emptyset$

```
Inti
BEGIN
    M :∈ N
END
```

```
    minimum
BEGIN
        act1 : m ∈min(ran(f))
END
```

5.2.2 First Refinement

Our first refinement consists in introducing, as in the previous example, two indices *p* and *q* where *p* is not greater than *q* as indicated in invariant inv1_3. Moreover, it is shown in invariant inv1_4 that the minimum of the array is in the set f [*p* .. *q*]:

Constants: n,f
Variables: m,p,q

Inv1-1: $p \in 1 \cdot\cdot n$
Inv2: $q \in 1 \cdot\cdot n$
Inv3: $p \leq q$

We also introduce two new events inc and dec. When p is smaller than q and f(p) is greater than f(q), we can reduce the interval p .. q to p + 1 .. q since f(p) is certainly not the minimum we are looking for. We have a similar effect with invariant dec. The minimum is then found when p is equal to q according to invariant inv1_4:

```
Init
m := 0
p:=1
```

```
Increment
When
p<q
f(p)>f(q)
then
p := p+1
end
```

```
decrement
When
p≤ q
f(p)>f(q)
then
q := q-1
end
```

```
minimum
When
p =q
then
m := f(p)
end
```

5.2.3 Pattern Design

First of all, in our notion, a pattern is just a development in Event-B including specification *p*0 and a refinement *p*1

5.2.3.1 Development Specification P0

Pattern design specification like initial model in the section 5.2.1

```
Constants : n, pat_f

Inti

pat_m:∈ ℕ
```

```
axm1  :  0 < n
axm2  :  Pat_f ∈ 1··n → ℕ
```

```
Min
BEGIN
    act1  :  Pat_m :=min(ran(pat_f))
```

5.2.3.2 Pattern refinement P1

Pattern refinement like first and first refinement in the section 5.2.2

```
Variables: pat_m , pat_p, pat_q
```

```
axm1  :  0 < n
axm2  :  Pat_f ∈ 1··n → ℕ
thm1  :  ran(pat_f)≠∅
```

```
inv1-1: pat_p ∈ 1··n
inv1-2: pat_q ∈ 1··n
inv1-3: pat_p ≤ pat_q
inv1-4: min(ran(f)) ∈ pat_f[pat_p··pat_q]
```

```
Inti
BEGIN
        act1 :pat_p := 1
        act2 :pat_q := n
```

```
Pat_Min
WHEN
        grd1  :  pat_p= pat_q
THEN
        act1  :  pat_m := pat_f(pat_p)
END
```

```
Pat_dec
WHEN
        grd1  :  pat_p<<pat_q
        grd2  :     pat_f(pat_p) ≤ pat_f(pat_q)
THEN
        act1  :  pat_q := pat_q-1
END
```

```
Pat_inc
WHEN
        grd    :  pa  _p< pat_q
        grd2  :      pa  _f(pat_p)> pat_f(pat_q)
THEN
        act1  :  pat_p := pat_p+1
END
```

Pattern design steps and applying SMT-solver of this model will be like that done in section 5.1.4 .3 and 5.1.5.

6. RESULT ANALYSIS

In section 5.1 we create the first case study that is a binary search model in Rodin platform including initial model and the first and second refinement. Rodin produces proof obligations for each model. Some of them are done automatically and the others need to discharge manually by the user or by proof solver tools as SMT-solver. The following table 1 shows the Proof statistics of Binary search model.

Table1. The Proof statistics of Binary search model.

Pos Methods / POs	**Total Pos**	**Auto**	**Interactive (SMT)**	**Proved Pos %**	**Saved Proof**	**Undercharged**
Rodin	37	10	0	10 (27%)	0	27
SMT	37	10	20	30 (81%)	0	7
Pattern	17	6	0	6 (35%)	20(54%)	11
SMT + Pattern	13	6	2	8 (61%)	24(64%)	5

When we create the binary search model in Rodin (section 5.1.1, 5.1.2, 5.1.3) we found 10 (27%) Pos produced automatically from total 37 Pos. The remaining Pos 27 need to discharge manually. Then we create the binary search model in Rodin by installing pattern plug-in (section 5.1.4). We found the required total proof 17 Pos instead of 37 Pos without pattern using that is mean the pattern approach saves 20 (54%)Pos instead of 37 Pos. But when we created the binary search model in Rodin by installing SMT-Solver plug-in as in section 5.1.5, we found 10 Pos produced automatically and 20 Pos produced by SMT-Solver from total 37 Pos. The remaining Pos 7 need to discharge manually that is mean using SMT-Solver approach the proof raised to (81%) proof instead of (27%) that obtained by applying Rodin only. Finally when we applied our proposal on binary search model in Rodin by installing SMT-Solver and pattern plug-ins together we found that the total proved jumps to 61% Pos compare with 27% Pos that is obtained from applying Rodin only, Also the saved proof jumps to 64% of total Pos. The following chart present different among Pos on Rodin, SMT-Solver, Pattern and SMT with Pattern binary search model.

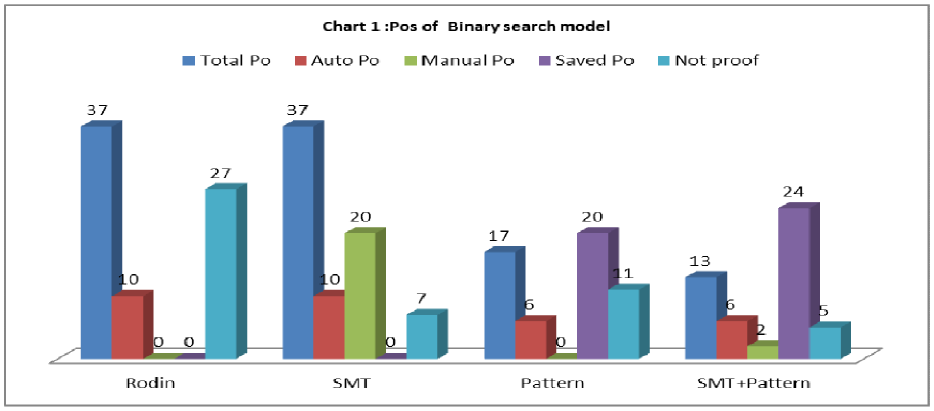


Chart 1: Pos of Binary search model

In section 5.2 we create the second case study that is a Minimum model in Rodin platform including initial model and the first refinement. Rodin produces proof obligation for each model. Some of them are done automatically and the others need to discharge manually by the user or by proof solver tools like SMT-solver. The following table 2 shows the Proof statistics of minimum search model.

Table 2: Proof statistics of Minimum model

Pos Methods / POs	**Total Pos**	**Auto**	**Interactive (SMT)**	**Proved Pos %**	**Saved Proof**	**Undercharged**
Rodin	25	5	0	5 (2%)	0	20
SMT	25	5	17	22 (88%)	0	3
Pattern	5	1	0	1 (2%)	20 (80%)	4
Smt + Pattern	4	1	2	3 (75%)	21 (84%)	1

When we create the minimum search model in Rodin (section 5.2.1, 5.2.2) we found 5 (2%) Pos produced automatically from total 25 Pos. The remaining Pos 20 need to discharge manually. Then we create the minimum search model in Rodin by installing pattern plug-in. We found the required total proof 5 Pos instead of 25 Pos without pattern using that is mean the pattern approach saves 20 (80%) Pos from 25 Pos. But when we create the minimum search model in Rodin by installing SMT-Solver plug-in, we found 5 Pos produced automatically and 17 Pos produced by SMT-Solver from total 25 Pos. The remaining Pos 3 need to discharge manually that is mean using SMT-Solver approach the proof raised to (88%) proof instead of (27%) that obtained by applying Rodin only. Finally when we applied our proposal on minimum search model in Rodin by installing SMT-Solver and pattern plug-ins together we found that the total proved jumps to 75% Pos compare with 2% Pos that is obtained from applying Rodin only, Also the saved proof jumps to 84% of total Pos. The following chart present different among Pos on Rodin, SMT-Solver, Pattern and SMT with Pattern of minimum search model.

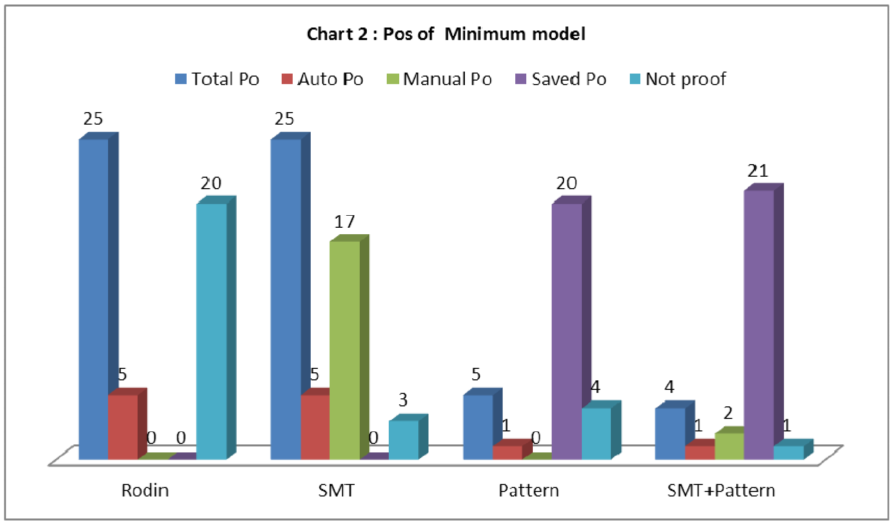


Chart 2: Pos of minimum model

7. Conclusion And Further Work

In the previous section we analyzed statistically and compare with among applying SMT-Solver and pattern or do not apply. Then we conclude that the proof obligation reduced than using Rodin only. Also when we applied SMT-Solver on pattern model that help for reusability without proof obligation again (saved Pos). Finally, when we present the results of the two different case studies, we found proposal reduced the proof obligation successfully. The method in this paper tends to reduce the proving effort, to reuse a model and to increase the degree of automation.

As for future work, Proof obligations can be discharged in other development platforms, such as Atelier-B. Also other proof solvers may be used to improve discharging the proof obligations. Furthermore, we intend to implement the missing features from the pattern plug-in for the RODIN Platform, e.g. syntax checking and support for extracting information from the pattern refinement. Also, we are going to investigate more examples in other domains that could benefit from pattern approach. However, we also need to "instantiate" the context of the pattern development. In our examples so far, the contexts of the pattern and the problem are the same. Also, we shall like to use the patterns in a more general context. In the future, if this turns out to be too restrictive, we can choose to generate the corresponding proof obligations, again for more flexibility. Note that if a pattern matching can be syntactically checked successfully, the proof obligations generated should be trivial to be discharged. Finally, we can apply Pattern approach with another proof solver and compare with our proposed method.

REFERENCES

[1] Abbasi, K. ; Akkaya, M. & Younis, A., (2007) "A distributed connectivity restoration algorithm in wireless sensor and actor networks", in: Proceedings of 32nd IEEE Conference on Local Computer Networks, LCN'07, IEEE, pp. 496–503.

[2] Abrial, J.-R & Hallerstede, S., (2006) " Refinement, De- composition and Instantiation of Discrete Models: Application to Event-B". Fundamentae Informatica.

[3] Abrial, J.-R, (1996) "The B-book: Assigning Programs to Meanings", Cambridge University Press, URL: http://portal.acm.org/citation.cfm?id=236705.

[4] Abrial, J.-R. ,(2007) "A system development process with Event-B and the RODIN platform", in: Proceedings of International Conference on Formal Engineering Methods, ICFEM'07, in: Lecture Notes in Computer Science, vol. 4789, Springer-Verlag, pp. 1–3.

[5] Abrial, J.-R. ; Butler, M. ; Hallerstede, S.; Hoang T.S., Mehta, F. &Voisin, L.: "RODIN: an open toolset for modelling and reasoning in event-B", Int. J. Softw. Tools Technol. Transf. 12(6), 447–466

[6] Abrial, J.-R.; Butler, M. ; Hallerstede, S.; Hoang, T.S. ; Mehta, F. & Voisin, L. , (2010) "Rodin: an open toolset for modelling and reasoning in Event-B", International Journal on Software Tools for Technology Transfer (STTT) (6) (2010) 447–466.

[7] Bobot,F.; Conchon, S.; Contejean, E. & Lescuyer, S., (2008), "Implementing Polymorphism in SMT solvers", in: C. Barrett, L. de Moura (Eds.), SMT 2008: 6th International Workshop on Satisfiability Modulo Theories.

[8] Bofill, M. ; Nieuwenhuis, R.; Oliveras, A. ; Rodrguez-Carbonell, E. & Rubio, A., (2008) "The barcelogic SMT solver", in: A. Gupta, S. Malik (Eds.), Proceedings of the 20th International Cconference on Computer Aided Verification, in: Lecture Notes in Computer Science, vol. 5123, Springer, Berlin,Heidelberg,pp.294–298.URL: http://www.springerlink.com/index/10.1007/978-3-540-70545-1.

[9] Bouton, T. ; de Oliveira, D.C.B. ; Déharbe, D. & Fontaine, P. , (2009) "veriT: an open, trustable and efficient SMT-solver", in: Automated Deduction—CADE-22, in: Lecture Notes in Computer Science, vol. 5663, Springer Verlag, pp. 151–156. URL: http://www.springerlink.com/content/f33m4615152325x3.

[10] Butler, M., (2009) " Decompostion Structures for Event-B". In: Integrated Formal Methods. Lecture Notes in Computer Science, vol.5423,pp.20–38.Springer,Berlin. http://www.springerlink.com

[11] Coleman, J. ; Jones, C. ; Oliver, I., Romanovsky, A. & Troubitsyna, E., (2005) "{RODIN} (Rigorous open Development Environment for Complex Systems)", in: Fifth European Dependable Computing Conference: EDCC-5 supplementary volume, , pp. 23–26.

[12] De Moura, L. & Bjr̈ner, N. , (2008) "Z3: an efficient SMT solver", in: C.R. Ramakrishnan, J. Rehof (Eds.), Proc. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, TACAS, in: Lecture Notes in Computer Science, vol. 4963, Springer, pp. 337–340.

[13] Déharbe,D.,(2011),"Integration of SMT-solvers in B and Event-B development environments ",Universidade Federal do Rio Grande do Norte, Departamento de Informática e Matemática Aplicada, Natal, RN, Brazil. [14]DEPLOY Project: (2009), "Deliverable JD1—Report on Knowledge Transfer". http://www.deploy-project.eu/pdf/ fv-d5-jd1-reportonknowledgetransfer.zip.

[15] Dutertre, B. & de Moura, L.,(2006)"The Yices SMT solver",Tool paper at http://yices.csl.sri.com/tool-paper.pdf.

[16] Fürst, A. & Hoang,T.S., (2010) “Rodin platform archive of question/response protocol”. http://deploy-eprints.ecs.soton.ac.uk

[17] Hoang, T. S. ; Fürst, A. & Abrial, J.-R. , (2010) 17 November © Springer-Verlag.

[18] Melodia, T. ; Pompili, D. ; Gungor, V.C. & Akyildiz, I.F., (2007) “Communication and coordination in wireless sensor and actor networks”, Proceedings of IEEE Transactions on Mobile Computing 6 (10) 1116–1129.

[19] The Eclipse Foundation, Eclipse SDK,(2009). URL: http://www.eclipse.org.

[20] Wieczorek, S., Roth, A., Stefanescu, A., Charfi, A., (2008)” Precise steps for choreography modeling for SOA validation and verification”, In: Proceedings of the Fourth IEEE International Symposium on Service-Oriented System Engineering. http://deploy-eprints.ecs.soton.ac.uk/41/

[21] Xiaohong Yuan, X. & Fernandez, E. , (2011)” PATTERNS FOR BUSINESS-TO-CONSUMER ECOMMERCE APPLICATIONS”, International Journal of Software Engineering & Applications (IJSEA), Vol.2, No.3, pp 1.