A Proof-Based Approach to Verifying Reachability Properties

Amel Mammar [1,2]

Institut Telecom SudParis
CNRS/SAMOVAR
Paris, France

Fama Diagne[3]

Institut Telecom SudParis
CNRS/SAMOVAR
Paris, France
GRIL, Département d'Informatique
Universite de Sherbrooke
Sherbrooke (Québec), Canada

Marc Frappier[4]

GRIL, Département d'Informatique
Universite de Sherbrooke
Sherbrooke (Québec), Canada

Abstract

This paper presents a formal approach to proving temporal reachability properties, expressed in CTL, on B systems. We are particularly interested in demonstrating that a system can reach a given state by executing a sequence of actions (or operation calls) called a path. Starting with a path, the proposed approach consists in calculating the proof obligations to discharge in order to prove that the path allows the system to evolve in order to verify the desired property. Since these proof obligations are expressed as first logic formulas without any temporal operator, they can be achieved using the prover of AtelierB. Our proposal is illustrated through a case study.

Keywords: Reachability properties, CTL formulas, B language, Proof obligations.

[1] This research is supported by the Natural Sciences and Engineering Research Council of Canada
[2] Email:amel.mammar@it-sudparis.eu
[3] Email:fama.diagne@it-sudparis.eu
[4] Email:marc.frappier@usherbrooke.ca

1 Introduction

Introduced by J.R Abrial [1], B is a formal method to developing safe systems. A "safe system" means a system that satisfies some safety properties and does not harm. To this aim, a B developer has to express such properties as invariants and specify the adequate conditions under which operations may be executed in order to maintain the desired properties. These conditions, called *preconditions*, aim at reducing the set of the allowed system behaviors to those that re-establish the invariants. Nevertheless, safety properties alone are insufficient as a basis of system design since they do not guarantee that the system would evolve and achieve its functionalities. Indeed, a system that stays infinitely in its initial state without performing any action is trivially safe but it becomes useless. That is why it is necessary to verify other kinds of properties called *liveness* properties [3].

In this paper, we are particularly interested in verifying reachability (subset of liveness) temporal properties, expressed in CTL [2], on a system specified in B. Such properties ensure that the system can evolve to reach a desired state. Basically, we suggest a formal approach to demonstrate that a path, defined with operation calls, allows the system to move into a state verifying a given property. The proposed approach consists in generating a set of proof obligations to ensure that the system does reach the desired state by following the specified path. The key idea is to store the values of all the state variables at each execution step of the path, prove that the current values allow the operation that follows to be executed, and establish that the system reaches the desired state by executing the last operation of the path. Since the proof obligations we derive are expressed as first-order logic formulas without any temporal operator, they can be discharged using the prover of AtelierB, for instance.

Section 2 presents the case study used in the sequel of the paper to illustrate the approach we propose in Section 3. Section 4 wraps up with the conclusion and future work.

2 Case study presentation

We illustrate our proposal with a simple library management system. The system has to manage book loans to members. Each member may either borrow or reserve a book if the latter is already lent to another member. Several reservations may be made on the same book. After doing a reservation on a book, a member borrows the book when the members, who made reservations before her/him, have already borrowed and returned the book or canceled their reservations. Also, we make the assumption that a member cannot borrow more than 2 books at the same time. The B specification corresponding to this system is as follows:

Machine *Library*
Sets *Books, Members*
Variables *books, members, loan, reserve*
Invariant $books \subseteq Books \wedge members \subseteq Members \wedge$
$loan \in books \nrightarrow members \wedge reserve \in books \rightarrow \mathbf{iseq}(members) \wedge$
$\forall me.\ (me \in members \Rightarrow \mathbf{card}(loan^{-1}[\{me\}] \leq 2)$
DEFINITIONS
/*$Index(bo, me)$ gives the rank of a member me in the reservation list of a book bo*/
Index(bo,me) $\widehat{=}\ (reservation(bo))^{-1}(me)$;
Operations
Lend(me , bo)$\widehat{=}$ **PRE** $me \in members \wedge b \in books \wedge bo \notin dom(loan) \wedge$
$(reservation(bo) = [] \textbf{ or } (\mathbf{first}(reservation(bo)) = me)) \wedge card(loan^{-1}[\{me\}]) < 2$
THEN
$loan := loan \lhd\!\!+ \{bo \mapsto me\}$
END;

Reserve(me, bo)$\widehat{=}$ **PRE** $me \in members \wedge bo \in books \wedge me \notin ran(reservation(bo)) \wedge$
$bo \mapsto me \notin loan \wedge ((bo \in dom(loan)) \textbf{ or } (reservation(bo) \neq []))$
THEN
$reservation := reservation \lhd\!\!+ \{bo \mapsto ((reservation(bo) \leftarrow me)\}$
END;

Return(bo)$\widehat{=}$ **PRE** $bo \in books \wedge bo \in dom(loan)$ **THEN**
$loan := \{bo\} \lhd\!\!\!- loan$
END;

Cancel_Reservation(me, bo)$\widehat{=}$
PRE $me \in members \wedge bo \in books \wedge me \in ran(reservation(bo))$
THEN
$reservation := reservation \lhd\!\!+$
$\{bo \mapsto ((reservation(bo) \uparrow (Index(bo, me) - 1)) \frown (reservation(bo) \downarrow Index(bo, me)))\}$
END
END

Using the prover of AtelierB, we have proved the *Library* specification by generating 13 proof obligations in order to ensure that the execution of each operation re-establishes the invariant: 11 of them have been discharged automatically while the others have required our intervention to help the prover find the right rules to apply. Nevertheless, such proof obligations do not guarantee that, for instance, a member *me* can always borrow a book *bo* expressed in CTL as follows:

$$\mathsf{AG}(bo \in books \wedge me \in members \wedge bo \mapsto me \notin loan) \Rightarrow \mathsf{EF}(bo \mapsto me \in loan)$$

The rest of the paper addresses such properties by defining the proof obligations that are sufficient to prove them.

3 Proving Reachability Properties

In this section, we describe the process we have defined to prove that a system verifies reachability properties expressed as CTL formulas of the form ($\mathsf{AG}\ \psi \Rightarrow \mathsf{EF}\ \phi$). Our proposal consists in demonstrating that starting from a state s verifying ψ, the system will reach a state s' where ϕ becomes true. To reach this new state s', the system has to execute a sequence of actions that we call a *path*. Recall that our objective is to generate the proof obligations that ensure that the system will reach the state s' following a conditional path of

the form $(cond \rightarrow (act_1 \,.\, \ldots \,.\, act_n))$ with " . " denoting the action sequencing, and notation $(cond \rightarrow Act)$ meaning that actions Act are executed if predicate $cond$ holds, otherwise nothing is done. More formally, we have to generate the proof obligations that establish the following predicate:

$$\psi \wedge cond \Rightarrow [(act_1 \,.\, \ldots \,.\, act_n)]\phi$$

One can then use traditional tools like AtelierB to prove it. However, the proof obligations generated can be huge, so hard to prove. Thus, we propose in this paper an other approach to proving it by generating more simple proof obligations. The key ideas of our approach can be summarized in the following three points:

(i) In state s where ψ is true and condition $cond$ holds, action act_1 can be executed, that is, its precondition holds,

(ii) After executing each action $act_{ii=1..(n-1)}$, the values of the machine variables verify the precondition of action act_{i+1},

(iii) After executing the last action act_n, the values of the different variables satisfy ϕ.

The actions we consider in this paper can be described according to the following BNF grammar:

$action$	::=	
	$op(param_1, \ldots, param_n)\|$	/*execution of operation op*/
	$\|_{v \in E}\ op(v, \ldots)\|$	/*execution of operation op for any value v in E*/
	$\mathsf{LOOP}(C, \|_{x \in E} op(x, \ldots), V)\|$	/*execution of operation op for each value of x of set E*/
	$cond \rightarrow action$	/*execution of an action if condition $cond$ is true*/

Action $\mathsf{LOOP}(C, |_{x \in E} op(x, \ldots), V)$ means that operation op is executed for any element of E while condition C is true. To ensure that this loop will eventually terminate, a strict natural decreasing expression V, called a *variant*, is specified.

According to this grammar and making the assumption that each operation is of the form (**PRE** P **THEN** S **END**), the precondition of an action is defined as follows:

Action	**Precondition**
$op(param_1, \ldots, param_n)$	P
$\|_{v \in E}\ op(v, \ldots)$	$\exists\, v.(v \in E \wedge P)$
$\mathsf{LOOP}(C, \|_{x \in E} op(x, \ldots), V)$	$C \Rightarrow \exists\, x.(x \in E \wedge P)$
$cond \rightarrow act$	$cond \Rightarrow Precondition(act)$

The following algorithm describes the generation of the proof obligations associated with a path. To facilitate the writing of the algorithm, we define the functions *update* and $Cons\_Po$ as follows:

$$Update(H, \forall X.P) = \begin{cases} \forall(Y,X).(P \wedge Q), & \text{if } (H = \forall Y.Q) \\ \forall X.P, & \text{otherwise} \end{cases}$$

$$Cons\_Po(H, P) = \begin{cases} \forall X.(Q \Rightarrow P), & \text{if } (H = \forall X.Q) \\ P, & \text{otherwise} \end{cases}$$

Algorithm Generation of proof obligations

Require: The CTL formula $F = (\mathsf{AG}\ \psi \Rightarrow \mathsf{EF}\ \phi)$ and a set of $paths = \{path_1, \ldots, path_m\}$

1: Let $POs = \emptyset$ be the set of the proof obligations to establish
2: **for each** $(path_p \in paths \wedge path_p \mathrel{\widehat{=}} (cond_p \rightarrow (act_1 \,.\, \ldots \,.\, act_n)))$ **do**
3: Let $Hyp = True$ be a set of local hypotheses
4: Let $For = \emptyset$ be the set of the formulas to prove
5: Let $V_{Spec} = \{x_1, \ldots, x_k\}$ be the set of the variables of the B specification
/*Dealing with action act_1*/
6: $For{=}For \cup \{Precond(act_1)\}$
7: **if** $(act_1 = \mathsf{LOOP}(C, |_{oo \in E}\ op(oo, \ldots), V))$ **then**
8: $For{=}For \cup \{Precond(op) \Rightarrow (V \in N \wedge [n := V][S](V < n))\}$
9: **else if** $(act_1 = |_{x \in E}\ \_)$ **then**
10: $Hyp = \forall x.(x \in E)$
11: **end if**
12: Let P be the before-after predicate of the substitution of act_1. This predicate, defined by $\neg([S](x' \neq x))$, expresses the value after ${x_i}^1$ of each variable x^i with respect to value before x_i
13: $Hyp = Update(Hyp, \forall({x_1}^1, \ldots, {x_k}^1).P)$
/*Dealing with the other actions*/
14: **for each** (action act_j such that $(j \in 2..n)$) **do**
15: $act_j = [\|x_i := {x_i}^{j-1}]act_j$
16: $For{=}For \cup \{Cons\_Po(Hyp, Precond(act_j))\}$
17: **if** $(act_j = \mathsf{LOOP}(C, |_{oo \in E}\ op(oo, \ldots), V))$ **then**
18: $For{=}For \cup \{Precond(op) \Rightarrow (V \in N \wedge [n := V][S](V < n))\}$
19: **else if** $(act_j = |_{x \in E}\ \_)$ **then**
20: $Hyp = Update(Hyp, \forall x.(x \in E))$
21: **end if**
22: Let P be the before-after predicate of the substitution of act_j. This predicate expresses the value after x_i^j of each variable x^i with respect to value before ${x_i}^{j-1}$
23: $Hyp = Update(Hyp, \forall({x_1}^j, \ldots, {x_k}^j).P)$
24: **end for**
25: $For{=}For \cup \{Cons\_Po(Hyp, [\|x_i := {x_i}^j]\phi)\}$
26: Let V_F be the set of variables of F
27: Let $V_F - V_{Spec} = \{y_1, \ldots, y_l\}$ be the set of variables of F that do not belong to Var_{Sepc}. These variables are free and are implicitly universally quantified
28: The proof obligation of $path_p$ is $PO(path_p) = \forall(y_1, \ldots, y_l).(\psi \wedge cond_p \Rightarrow (\bigwedge_{for \in For} for))$
29: **end for**
30: $POs = \bigcup_{i=1,m} PO(path_i) \cup \{\forall(y_1, \ldots, y_l).(\psi \Rightarrow \bigvee_{i=1..m} cond_i)\}$

Let us give some explanations about the above algorithm:

- The proof obligation $(Precond(op) \Rightarrow (V \in N \wedge [n := V][S](V < n)))$ (line 8) allows to prove that the loop will eventually terminate.

- Statement $(act_j = [\|x_i := {x_i}^{j-1}]act_j)$ (line 15) permits to replace the value of each variable x_i with its current value ${x_i}^{j-1}$ produced by the execution of action act_{j-1}.
- The additional proof obligation generated in line 30 ensures that the paths $\{path_1, \ldots, path_m\}$ are sufficient to cover, in state s (where ψ is true), all the possible values of the variables V_{Spec}.

Let us illustrate the above algorithm on the case study of section 2 for the following CTL formula and paths:

- $F \mathrel{\widehat{=}} \mathsf{AG}(me \in members \wedge bo \in books \wedge bo \mapsto me \notin loan) \Rightarrow \mathsf{EF}(bo \mapsto me \in loan)$
- $$\begin{aligned} path_1 \mathrel{\widehat{=}}\ & reservation(bo) = [] \rightarrow \\ & (\\ & (bo \in dom(loan) \rightarrow Return(bo)). \\ & ((card(loan^{-1}[\{me\}]) = 2) \rightarrow |_{bo' \in loan^{-1}[\{me\}]} Return(bo')). \\ & Lend(me, bo) \\ &) \end{aligned}$$
- $$\begin{aligned} path_2 \mathrel{\widehat{=}}\ & reservation(bo) \neq [] \rightarrow \\ & (\\ & (me \notin ran(reservation(bo)) \rightarrow Reserve(me, bo)). \\ & \mathsf{LOOP}(reservation(bo) \neq [me], |_{oo \in ran(reservation(bo)) - \{me\}} Cancel(oo, bo), \\ & \qquad \mathbf{size}(ran(reservation(bo)))). \\ & (bo \in dom(loan) \rightarrow Return(bo)). \\ & ((card(loan^{-1}[\{me\}]) = 2) \rightarrow |_{bo' \in loan^{-1}[\{me\}]} Return(bo')). \\ & Lend(me, bo) \\ &) \end{aligned}$$

Applying the previous algorithm on $path_1$ gives the following results [5]:

- $act_1 = (bo \in dom(loan) \rightarrow Return(bo))$

$$For \mathrel{\widehat{=}} \{bo \in dom(loan) \Rightarrow (bo \in books \wedge bo \in dom(loan))\}$$

$$\begin{aligned} Hyp \mathrel{\widehat{=}}\ & \forall(,_{v_1 \in V_{Sepc}}).(\\ & ((bo \in dom(loan) \wedge loan_1 = \{bo\} \lhd loan) \textbf{ or} \\ & (bo \notin dom(loan) \wedge loan_1 = loan)) \wedge \textstyle\bigwedge_{v \in V_{Spec} - \{loan\}} (v_1 = v)) \end{aligned}$$

[5] Notation $,_{v \in \{x_1 \ldots, x_n\}}$ denotes $x_1, \ldots, x_n$.

- $act_2 = ((card(loan^{-1}[\{me\}]) = 2) \rightarrow |_{bo' \in loan^{-1}[\{me\}]} Return(bo'))$

$$
\begin{aligned}
For \mathrel{\widehat{=}}\ & For \cup \{\forall(,_{v_1 \in V_{Sepc}}).(\\
& ((bo \in dom(loan) \wedge loan_1 = \{bo\} \lhd loan) \textbf{ or} \\
& (bo \notin dom(loan) \wedge loan_1 = loan)) \wedge \textstyle\bigwedge_{v \in V_{Spec} - \{loan\}} (v_1 = v) \Rightarrow \\
& (card({loan_1}^{-1}[\{me\}]) = 2 \Rightarrow \\
& \qquad \exists\, bo'.(bo' \in {loan_1}^{-1}[\{me\}] \wedge bo' \in books_1 \wedge bo' \in dom(loan_1)))))\}
\end{aligned}
$$

$$
\begin{aligned}
Hyp \mathrel{\widehat{=}}\ & \forall(,_{v_1 \in V_{Sepc}},\ ,_{v_2 \in V_{Sepc}}, bo').(((bo \in dom(loan) \wedge loan_1 = \{bo\} \lhd loan) \textbf{ or} \\
& (bo \notin dom(loan) \wedge loan_1 = loan)) \wedge \textstyle\bigwedge_{v \in V_{Spec} - \{loan\}} (v_1 = v) \wedge \\
& ((card({loan_1}^{-1}[\{me\}]) = 2 \wedge loan_2 = \{bo'\} \lhd loan_1) \textbf{ or} \\
& \qquad (card({loan_1}^{-1}[\{me\}]) \neq 2 \wedge loan_2 = loan_1)) \wedge \\
& \textstyle\bigwedge_{v \in V_{Spec} - \{loan\}} (v_2 = v_1) \wedge \\
& (bo' \in {loan_1}^{-1}[\{me\}] \wedge bo' \in books_1 \wedge bo' \in dom(loan_1)))
\end{aligned}
$$

- $act_3 = Lend(me, bo)$

$$
\begin{aligned}
For \mathrel{\widehat{=}}\ & For \cup \{\forall(,_{v_1 \in V_{Sepc}},\ ,_{v_2 \in V_{Sepc}}, bo').(\\
& ((bo \in dom(loan) \wedge loan_1 = \{bo\} \lhd loan) \textbf{ or} \\
& (bo \notin dom(loan) \wedge loan_1 = loan)) \wedge \textstyle\bigwedge_{v \in V_{Spec} - \{loan\}} (v_1 = v) \wedge \\
& ((card({loan_1}^{-1}[\{me\}]) = 2 \wedge loan_2 = \{bo'\} \lhd loan_1) \textbf{ or} \\
& \qquad (card({loan_1}^{-1}[\{me\}]) \neq 2 \wedge loan_2 = loan_1)) \wedge \\
& \textstyle\bigwedge_{v \in V_{Spec} - \{loan\}} (v_2 = v_1) \wedge \\
& (bo' \in {loan_1}^{-1}[\{me\}] \wedge bo' \in books_1 \wedge bo' \in dom(loan_1)) \Rightarrow \\
& (me \in members_2 \wedge bo \in books_2 \wedge bo \notin dom(loan_2) \wedge \\
& (reservation_2(bo) = [] \textbf{ or } (\textbf{first}(reservation_2(bo)) = me)) \wedge \\
& card({loan_2}^{-1}[\{me\}]) < 2)\}
\end{aligned}
$$

$$
\begin{aligned}
Hyp \mathrel{\widehat{=}}\ & \forall(,_{v_1 \text{`} \in V_{Sepc}}, ,_{v_2 \in V_{Sepc}}, bo',\ ,_{v_3 \in V_{Sepc}}).(\\
& ((bo \in dom(loan) \wedge loan_1 = \{bo\} \lhd loan) \textbf{ or} \\
& (bo \notin dom(loan) \wedge loan_1 = loan)) \wedge \\
& \textstyle\bigwedge_{v \in V_{Spec} - \{loan\}} (v_1 = v) \wedge \\
& ((card(loan_1\ [me]) = 2 \wedge loan_2 = \{bo'\} \lhd loan_1) \textbf{ or} \\
& \qquad (card(loan_1\ [me]) \neq 2 \wedge loan_2 = loan_1)) \wedge \\
& \textstyle\bigwedge_{v \in V_{Spec} - \{loan\}} (v_2 = v_1) \wedge \\
& (bo' \in {loan_1}^{-1}[\{me\}] \wedge bo' \in books_1 \wedge bo' \in dom(loan_1)) \\
& loan_3 = loan_2 \cup \{bo \mapsto me\} \wedge \textstyle\bigwedge_{v \in V_{Spec} - \{loan\}} (v_3 = v_2))
\end{aligned}
$$

Putting $(Hy \mathrel{\widehat{=}} \forall(,_{v_1 \text{`} \in V_{Sepc}}, ,_{v_2 \in V_{Sepc}}, bo',\ ,_{v_3 \in V_{Sepc}}).P)$, we obtain:

$$For \mathrel{\widehat{=}} For \cup \{\forall(,_{v_1 \text{`} \in V_{Sepc}}, ,_{v_2 \in V_{Sepc}}, bo',\ ,_{v_3 \in V_{Sepc}}).(P \Rightarrow (bo \mapsto me \in loan_3)\}$$

Finally to prove that $path_1$ allows to reach a state where $(bo \mapsto me \in loan)$, we have to establish:

$$\forall(bo, me).(bo \in book \wedge me \in member \wedge bo \mapsto me \notin loan \wedge reservation(bo) = [] \Rightarrow \bigwedge_{for \in For} for)$$

Using AtelierB, we have added these proof obligations as assertions (clause **ASSERTIONS** of B) to the B machine of page 2. These assertions generate 7 PO: 5 are automatically proved and 2 are easily proved with the interactive prover.

4 Conclusion

In this paper, we have presented a formal approach to verifying reachability properties expressed in CTL on B systems. The main idea of this process consists in identifying the possible paths that the system has to follow to reach the desired state and automatically generating B proof obligations to ensure that these paths permit to verify the CTL properties.

Some work has been done on specifying and proving liveness properties using Event B. In [4], an approach to specifying and verifying liveness properties on B Event systems is described. The liveness properties, specified using UNITY, are integrated to the Event B specification in order to be proved. The authors deals with properties of the form: *if predicate P ever becomes true, then the system state will change eventually to another state where predicate Q is true.* A similar approach using the TLA+ language is proposed in [5]. These approaches deal with liveness properties that are stronger than our reachability properties that aim at exhibiting at least one execution starting from a state satisfying P to reach another state where Q becomes true.

In [6], we have also proposed an other approach to prove reachability properties using the refinement calculus of Carroll Morgan [7] in a B context. Compared to the approach presented here, the proof obligations it generates are more simple but the generation process requires more information from the user.

Future work includes the correctness proof of the algorithm we have developed and the automation of the proposed approach. We plan also to consider other reachability property forms.

References

[1] Abrial, J.R.: The B-Book: Assigning Programs to Meanings, Cambridge University Press, (1996).

[2] Clarke, E.M., Emerson E.A.: Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic", in Logic of Programs Workshop. LNCS 131, Springer-Verlag (1982).

[3] Lamport, L.: Proving the Correctness of Multiprocess Programs, IEEE Transactions on Software Engineering, Volume 3 , Issue 2, (1977).

[4] Barradas, H.R., Bert, D.: Specification and Proof of Liveness Properties under Fairness Assumptions in B Event Systems Workshop on Algebraic Development Techniques, (2004).

[5] Mosbahi,O., Jaray J.: Specification and Proof of Liveness properties in B Event System International Conference on Software and Data Technologies, (2007).

[6] Frappier, Diagne, F., Mammar, A.: Proving Reachability in B using Substitution Refinement Submitted to the B Dissemination Workshop, (2010).

[7] Morgan, C.C.: Programming from Specifications, Second edition, Prentice Hall, (1998).