Proof and Model-checking, two complementary Approaches

Henri Habrias
I.U.T, département informatique
Université de Nantes
Nantes, France

Abstract
We show with a small example, a two-places buffer, how proof, refinement and model-checking can be used in a complementary manner. Our software tools are Atelier B and LTSA.

Keywords: Formal specifications, Teaching, B method, Model-checking

"Thus some people do not listen to a speaker unless he speaks mathematically, others unless he gives instances, while others expect him to cite a poet as witness. And some want to have everything done accurately, while others are annoyed by accuracy, either because they cannot follow the connexion of thought or because they regard it as pettifoggery." (Aristotle, Metaphysics, Book 1 Part 3)

1 Introduction

It exists different paradigms and specification languages. A presentation through a same case study is done in [5]. We quote [8]: "We can distinguish two verification approaches: - One putting into practice an inference system, i.e. a well defined and formalised set of reasoning rules issued of the logic (modus ponens, recurrence, equal values substitution, etc.); the support tools are proof assistants (PVS, HOL, Coq, AtelierB, LP,...) - The other is based on the construction of a model generally finite representing the transitions system representing the behaviour of the modelised application. The verification of the properties is done by an exhaustive exploration of this model. The support tools are based on the principle of the evaluation of the properties on a model, known under the name of model checking (MEC, CAESAR-Aldebaran, Auto, Spin, Cospan, SMV,...)"

1 The exercice on the buffer is taken from [3]. We want to thank Steve Dunne and Frank Zeida, from the University of Teesside who presented this exercise during their visits in IUT de Nantes on Erasmus exchanges.

2 Email: henri.habrias@univ-nantes.fr
We show in this paper, using an example, how to use the two approaches: the proof and model-checking to verify (check) a specification. We put them into practice using two tools: Atelier B and LTSA [6]. This example is used in our teaching for first year students in computer science [5]. Today the functionalities of these tools are available under the form of plug-in for Eclipse (for B, see Projet Rodin: www.inria.fr/recherche/equipes/rodin.fr.html). We expose our small example, and then we deal with it with the B method and Atelier-B (using the proof) and with the FSP language and the LTSA workshop (using model-checking). The use of ProB is presented in another paper of this conference. A presentation of the two specification approaches comes before their use.

2 The two places buffer

We start this study considering the behaviour of a two places buffer. We do not take into account the data going through this buffer. We give to the students some observations of traces:

\[ [in, out, in, out] \]
\[ [in, in, out, in, out, in, out] \]

And we precise that the following traces are never observed:

\[ [in, in, in] \]
\[ [out, in, out] \]
\[ [in, in, out, out, out] \]

Using the notation for the regular expressions, we can modelized the behaviour of the buffer in such a manner:

\[ M = (in.(in)^* out)^* \]

We can also use the Jackson trees (Fig: 1) which have the advantage to permit the naming of the nodes and the adding of information on the leaves.

3 Using the B method

We do not detail the B method here ([1], [2]).

3.1 Classical B vs Event B

In event B, it is very convenient to construct the automaton and to refine it (the Harel automata are very practical for that) and at the same time to construct the B specification and its refinement.

Very often the operations of classical B (Figure 2) are not understood by the students. They use preconditions stronger than what the invariant necessitates to be respected. A symptomatic example is the one of the door. Even if they know that if the before state is state = closed then state := closed respects the invariant
state \in \{\text{closed, open}\}$, they try to specify that is specified in Event B. So when we present classical B, we wait that some student said that our specification is bad, to speak about Event B (Figure 4).

We use the figures 3,5 to show that in Event B, we are considering the observation of a closed system. Nobody (excuses for E. Dijkstra \footnote{“The use of anthropomorphic terminology when dealing with computing systems is a symptom of professional immaturity”, E.W. Dijkstra}) asks an event to be executed! An event occurs and the occurring of an event can be observed. A specified event (type) may occur. On the contrary somebody can ask to do an operation even an operation doing nothing. We have to pay attention to our discourse to the students: somebody can observe what the call of an operation does. If we call the operation open when the doorState is open, we can observe that the operation has been called and that the operation did not change the state of the door. So we concluded that it is better to say to the students in a first approach that an event occurs and an operation is called, than to say that an event is observed.

3.2 The parallel composition in B and the diamond

The parallel composition of two processes uses more often what we call the diamond semantic: to execute the sequence $P_1; P_2$ gives the same result than to execute the sequence $P_2; P_1$. With the definition of the parallel composition used in B, it is not necessary the case as it is shown on the following example. Considering the state $x = 3 \land y = 1$ before the composition of the two operations $op_1 \triangleq x := x + 2$ and $op_2 \triangleq y := x + 2$.

In B, the execution of these two operations composed in parallel ($op_1 \parallel op_2$) gives the after state $x = 5 \land y = 5$.

The execution of these two operations composed in sequence in such a manner $x := x + 2; y := x + 2$, gives the following after state $x = 5 \land y = 7$. The execution of
MACHINE
  ClassicalDoor
SETS
  STATE = \{open, closed\}
VARIABLES
  doorState
INARIANT
  doorState ∈ STATE
INITIALISATION
  doorState := ETAT
OPERATIONS
  open  ≡ doorState := open END;
  fermer ≡ doorState := closed END;
  st ← whatState ≡ st := doorStateEND;
END

Fig. 2. The door in classical B

... (diagram of a door in classical B)...

these two operations composed in sequence in such a manner \( y := x + 2; x := x + 2 \), gives the following after state \( x = 5 \land y = 5 \). The diamond semantic is not respected. The two sequences do not give the same state.

3.3 The buffer with the B method

3.3.1 A first solution

We modelise the buffer by a machine \( M1 \) (Fig. 9) which encapsules a variable giving the number of data in the buffer. We remark that we do not modelise the values of these data. The guards are modelised in B by a SELECT. Now, we analyse the buffer and it appears that it is composed of two cells (see Figure 6). One is considered as feeding the second. We name the first PROD and the second CONS.

For each one, we specify an abstract machine (Fig. 7, 8).

These two machines are included in a third machine \( M2 \) (Fig. 12). An including machine can use in the definition of its operations, the operations of the included machines. But the operations of the included machines are not part of the repertoire of the operations of the including machine. Nevertheless, it is possible to promote operations of the included machines, operations then becoming elements of the repertoire of the operations of the including machine. It is what is done in the machine \( M2 \) (Fig. 12). Every one of these operations being proved, we specify an
MACHINE
  EventDoor
SETS
  STATE = \{open, closed\}
VARIABLES
  doorState
INARIANT
  doorState \in STATE
INITIALISATION
  doorState :\in STATE
OPERATIONS
  opening \triangleq
    \text{SELECT} \ doorState = closed \ \text{THEN} \ doorState := open \ \text{END};
  closing \triangleq \text{SELECT} \ doorState = open \ \text{THEN} \ doorState := closed \ \text{END};
  st \leftarrow \text{whatState} \triangleq \ st := \ doorState \ \text{END}
END

Fig. 4. The door in Event B

Fig. 5. The door in Event B

Fig. 6. The two places buffer

implementation machine which will permit us to prove that $M_2$ simulates $M_1$.

3.3.2 An other solution
We give a variante of the first solution.

4 Use of LTSA
4.1 Labelled transitions system and LTSA
The language $FSP$ of LTSA is very simple. In the following we give the translation into $FSP$ of some specifications written in the language of regular expressions.

4.2 From regular expressions to FSP
$P = aa^*$
MACHINE
	PROD
VARIABLES
	pstate
INARIANT
	pstate ∈ 0..1
INITIALISATION
	pstate := 0
OPERATIONS
	in ≡
		SELECT pstate = 0 THEN
			pstate := 1 END;
	pmove ≡
		SELECT pstate = 1 THEN
			pstate := 0 END
END

Fig. 7. Machine Producer

MACHINE
	CONS
VARIABLES
	cstate
INARIANT

cstate ∈ 0..1
INITIALISATION

cstate := 0
OPERATIONS
	out ≡ SELECT cstate = 1
	THEN cstate := 0
	END;
cmove ≡
	SELECT cstate = 0 THEN
	cstate := 1
END

END

Fig. 8. Machine Consumer

P = (a -> STOP | a -> STATE),
STATE = (a -> STATE).

P = (a, b, c)*

P = (a -> P | b -> P | c -> P).
P = (a(cd)*, b(cd)*)*
MACHINE M1
VARIABLES state
IN Variant
state \in 0..2 /* The state is the number of data in the buffer */
INITIALisation
state := 0
OPERATIONS
in \triangleq \text{select} state = 0 \text{ then } state := 1
\quad \text{when } state = 1 \text{ then } state := 2 \text{ end;}
out \triangleq \text{select} state = 1 \text{ then } state := 0
\quad \text{when } state = 2 \text{ then } state := 1 \text{ end;}
move \triangleq \text{skip}
END

Fig. 9. Machine M1

IMPLEMENTATION M1_I
* machine to prove that M2 simulates M1 *
REFINES M1
IMPORTS M2
PROMOTES in, out, move
END

Fig. 10. Implementation of M1

MACHINE M1
VARIABLES
NumberofData
IN Variant
NumberofData \in 0..2
INITIALisation
NumberofData := 0
OPERATIONS
in \triangleq \text{select} NumberofData < 2
\quad \text{then } NumberofData := NumberofData + 1 \text{ end;}
out \triangleq \text{select} NumberofData > 1
\quad \text{then } NumberofData := NumberofData − 1 \text{ end}
END

Fig. 11. Another solution, machine M1

\begin{align*}
P &= (a \rightarrow \text{STATE1} \mid b \rightarrow \text{STATE1}), \\
\text{STATE1} &= (a \rightarrow \text{STATE1} \mid b \rightarrow \text{STATE1} \mid c \rightarrow \text{STATE2}),
\end{align*}
Fig. 12. Another solution: $M_2$ refinement of $M_1$

\[
\text{STATE}_2 = (d \rightarrow \text{STATE}_1).
\]

$FSP$ is less powerful than the language of the regular expressions. Indeed, we cannot specify the case of the empty sequence. $P=a^*$ cannot be expressed in $FSP$.

### 4.2.1 Communication in $FSP$

$FSP$ is very close to $CCS$ of Milner ([7]). Whereas the communication between two processes in $CCS$ is done by complementary actions - we note them by a prime - , in $FSP$ it is done by actions (or events) of same name. So the following $CCS$ specification

\[
C_1 = \text{in.m.} C_1
\]
\[
C_2 = m'.\text{out}.C_2
\]
\[
\text{System} = C_1 \parallel C_2
\]

is written in $LTSA$:

\[
C_1 = (\text{in} \rightarrow m \rightarrow C_1).
\]
\[
C_2 = (m \rightarrow \text{out} \rightarrow C_1).
\]
\[
\text{SYSTEM} = (C_1 \parallel C_2).
\]

### 4.3 The absence of variables in $FSP$

We can qualify $FSP$ of pure as we qualify $CCS$ of pure relatively to ”value passing $CCS$”. In $FSP$, as in pure $CCS$, a variable is seen as a process. By example, a counter whose the values belong to the set $0..3$ will be modelised in such a manner:
ZERO = (plus -> ONE),
ONE = (minus -> ZERO | plus -> TWO),
TWO = (minus -> UN | plus -> THREE),
THREE = (minus -> TWO).

If we want to modelise the input values, we have to consider as much channels as considered values. So a one-place buffer which can receive as input values, values belonging to the set 0..3 can be specified in FSP in such a manner:

range T=0..3

4.3.1 The guard in FSP
A guarded event is specified in FSP by a when.

4.3.2 From sequence diagrams to the labelled transitions system
MSC (Messages Sequences Charts) have been defined with an abstract semantic in terms of transitions system and of parallel composition. A tool [9] permits to translate a specification of scenarios into diagrams of states transitions. LTSA permits to detect the presence of "non implied scenarios". We do not deals with this aspect here.

4.4 Specification of the buffer
The figure 13 illustrates this specification.

Fig. 13. The two places buffer, the parallel composition and the $\tau$.

4.4.1 Specification of high level
BUFFER = STATE0,
STATE0 = (in -> STATE1),
STATE1 = (in -> STATE2 | out -> STATE0),
STATE2 = (out -> STATE1).
BUFFER(N=2) = STATE[0],
STATE[i:0..N] = (when (i<N) in -> STATE[i+1] |
    when (i>0) out -> STATE[i-1]).

4.4.2 Specification of low level
PROD = (in -> move -> PROD).
CONS = (move -> out -> CONS).
||BUFF_2 = (PROD || CONS).

If now we hide the silent action, the move, this action is replaced by a \( \tau \). If we ask the minimisation of the automaton, we obtain then the same automaton as the one of the high level specification. So we have an equivalence between the two behaviours. In FSP, for the minimisation, two equivalences are considered. The strong equivalence considers that two systems are equal if they have the same behaviour when the occurrence of all their actions can be observed included the occurrence of the silent action. The minimisation uses this equivalence when there is no silent action. The weak equivalence considers that two systems are equal if they exhibit the same behaviour for an external observer who cannot detect the occurrence of the \( \tau \) actions.

PROD = (in -> move -> PROD).
CONS = (move -> out -> CONS).
||BUFF_2 = (PROD || CONS)\{move\}.

LTSA permits also to verify safety properties, as the absence of deadlock. A safety property defines a deterministic process asserting that this same process accepts every trace including the actions of the alphabet of this process. It permits also to verify the liveness properties. Such properties assert that something good will arrive eventually. In FSP, the temporal logic is not used and it is interesting for me because I have no time teach it. In FSP we are limited to a class of liveness property called progress, which is the opposite to the property of starvation. Such a property asserts that, whatever will be the state of the system, it is always the case where a specified action will eventually be executed.

5 Conclusion

In this short paper, we wanted to demonstrate how two approaches can be used to specify, refine and verify software. The B approach permits to manipulate variables as it is done classically in programming. The approach by Process Algebras as CCS does not use variables. A state is modelised by a behaviour. It is the possible behaviour when we are in this state. A state is an agent behaviour, following the vocabulary of Milner \cite{Milner73}. Some people can say that the approach is more abstract than the B approach, other people, on the contrary, considering that naming the states is an abstraction activity that is not useful in the process approach, can say that the approach is less abstract. It exists process algebras which use variables (they are named mixed approaches). Thanks to LTSA, we are obliged to modelise a variable by a behaviour, and it complicates the specification comparatively to what will be done with B. But otherwise with B the automaton is not explicit whereas LTSA products the automaton. Nevertheless the experience shows that very often to construct the specification in FSP or in CCS, we use the drawing of a transition diagram (see Fig. 14) which helps to structure the specification in FSP. So the two approaches are complementary. In B we are guided by the proof, in FSP we are guided by the execution generated by the tool and also by the automaton generated and his minimisation. It is possible, considering the automaton, restructurate the
Habrias

specification in FSP. With ProB, we have a tool which is made to work from a B specification. It gives us much more functionalities than LTSA and allows us to make model-checking before the proof. When a proof fails we can see the reason why it failed using model-checking. So the proof and the model-checking techniques as those of refinement in event B and those of refinement by decomposition in processes are well complementary.

\[
\text{Marriage(N:=N+1)} \quad \text{Divorce or death/}
\]
\[
\text{SELECT N=1 THEN N:=N-1}
\]

Bachelor Married

Marriage(N:=N+1)

Divorce or death/ SELECT N=1 THEN N:=N-1

Fig. 14. Polygamy without polyandry, Harel Automata with guards for men

Our teaching is divided into two "modules". In the first one, we emphasize on the static aspects of the specification. Classical B and the n-ary relational model are used. In the second one, we emphasize on the dynamic aspects and FSP and Event-B are used. The connection between the two is done by exercises as the following.

We ask the students to modelise the different types of conjugal relationships using B or the notation of Codd for the n-ary relational model, and using automata and FSP. We show that classical B is not sufficient to represent the different cases. Here is an example of a question of one of our examinations:

Consider the following specification written in FSP. By MEN we want to denote "the behaviour" of the physical persons of sex masculine. To avoid an combinatorial explosion of the number of states, we limited to 2 the number of spouses in course. To simplify, we do not have considered the death of a man who was never married. So our specification is not a good model of the reality.

\[
\text{MEN(N=3) = BACHELOR[0],}\]
\[
\text{BACHELOR[spouses_nb : 0..N] = (marriage -> MARRIED[1]),}\]
\[
\text{MARRIED[spouses_nb : 0..N] = (when (spouses_nb == 1) divorce -> MEN |}\]
\[
\text{when (spouses_nb > 1) divorce -> MARRIED[spouses_nb -1] |}\]
\[
\text{when (spouses_nb == 3) it_is_enough_for_a_man ->}\]
\[
\text{divorce -> MARRIED[spouses_nb - 1] |}\]
\[
\text{when (spouses_nb < 3) marriage -> MARRIED[spouses_nb + 1]).}\]

Say what are the following relational schemas (without null values) that are not in contradiction with this specification?

• Schema 1
  Mariages_in_course (id_married_men, id_married_women)

• Schema 2
  Mariages_in_course (id_married_men, id_married_women)

• Schema 3
  Mariages_in_course (id_married_men, id_married_women)
• Schema 4
Mariages_in_course (id_married_men, id_married_women)

For every above schema, what B specification says the same thing?

```
SETS
PERSON
VARIABLES
men, women
INVARIANT
men ⊆ PERSON ∧
women ⊆ PERSON ∧
* B1 *
isTodayMarriedWith ∈ married_women → married_men
* B2 *
isTodayMarriedWith ∈ married_men → married_women
* B3 *
isTodayMarriedWith ∈ married_women ← married_men
* B4 *
isTodayMarriedWith ∈ married_women ⇔ married_men
∧ dom(isTodayMarriedWith) = married_women
∧ ran(isTodayMarriedWith) = married_men
```

The answer is schemas 1 and 4.

References