Practical Combination of Theorem Proving and Model Checking for the Multi-facet Analysis: a Case Study

J. Christian Attiogbé

LINA - FRE CNRS 2729 - University of Nantes - France
Christian.Attiogbe@lina.univ-nantes.fr

In Proceedings SOFSEM'05, ISBN 80-969255-4-7

Abstract. Theorem proving and model checking are recognized as formal analysis techniques used to mechanize formal methods of software development. However the tools based on them are not widely and systematically used as they should. One reason for this is the lack of practical approaches to guide the users. Another reason is that the tools often focus on one specific aspect of analysis. Tool-assisted analysis of software systems and convenient guidance to practise formal methods are still motivating challenges. This paper addresses these challenges and shows using a case study, how one can combine both analysis techniques.

1 Introduction

We present a stepwise approach that can be systematically (re)used during the formal analysis and development where both theorem proving and model checking techniques are needed. Very often these techniques require some specific input languages or models. This may conduct to misleading results if there is no insurance that the starting models are equivalent. Moreover it is a burden for the developers. The contribution of this paper is an approach which shows that a *abstract reference model* may be the basis of forthcoming specific models and the basis of both analysis tracks, even if their input languages are different. The proposed analysis approach consists of building a general reference model of the system at hand, to derive systematically the specific models of each involved analysis technique from the reference model, to perform the analysis directly on the derived specific models or to extend them and then to perform the analysis on the extended specific models. The result of each part of the analysis may help to tune the reference model. In this paper we adopt a reference model described with logical rules. The B method [1] and the Spin system [2,3] are respectively chosen for theorem proving and for model checking. The choice of the B method is motivated by the fact that it provides both theorem proving (for safety properties) and refinement to code. The Spin system is chosen because it has a powerful model checking tool (for liveness properties) and it provides a user-friendly interface. Some of the existing related works use the high automation of model checking to master the lack of guidance of theorem proving [4]. Others focus on

the extension of model checking with theorem proving so as to decompose large problems into smaller ones[5]. In constrast, our work concentrates on the use of a single model and the complementary use of both techniques with respect to safety and liveness properties verification.

The paper is organized as follows. Section 2 presents the cooling process and its supervision system used as a working example. Section 3 shows how to build a reference model for the multi-facet analysis. Section 4 and Section 5 deal respectively with the derivation of specific models devoted to safety analysis facet and the liveness analysis facet. Section 6 concludes the paper.

2 The Cooling Process and an Initial Model

The system to be specified and analysed has to supervise a cooling process: the cooling of a liquid circulating with a constant delivery, by using water. The system is taken from [6] where the authors give a specification in the form of a logical rule-oriented modelling. We refer to this latter as an initial model. The cooling process is made of two parts: the first one is a water reservoir with the associated feed pump (*pump2*) and a floodgate (*fg2*); the second part is the cooling process itself; it uses water from the reservoir and makes it circulate to cool the fluid. This part is equipped with a floodgate (*fg1*) connected to the reservoir, followed by a circulation pump (*pump1*). The cooling system is a control system which supervises both parts. However a particularity of the current cooling system is that it doesn't avoid *bad states* (those in which malfunctions are detected) but it detects the bad states and it recommends to the human operator the adequate operations to correct them. The cooling system interacts with a human operator by indicating the state of the process and the recommended operations (orders) to be performed. All the recommended operations are not executed by the operator. He/She can decide which one is important and can give *execution orders* to the system. A supervision part is about the water reservoir level. The system should adopt an appropriate reaction if the water level is out of some given thresholds. Two all-or-none sensors[1] (`low-wlvl, high-wlvl`) are used to detect respectively "too low water level" and "too high water level" of the reservoir. The other part of the supervision is about the temperature of the liquid which is being cooled. Two all-or-none sensors (`low-temp, high-temp`) are used to detect the overstepping of the liquid low and high temperature. The system should help the operator to maintain the temperature between two given thresholds. If the temperature is too high the circulation pump *must be engaged*; if the temperature is too low, the circulation pump *must be disengaged*. This corresponds to the recommended operations. The operator can give orders to the cooling system from two control panels: a main panel (MP) and a rescue panel (RP). A pump engagement order (eo) or a disengagement order (do) can be given from a panel under certain conditions (too high liquid temperature, lack of water in the reservoir, etc).

[1] An all-or-none sensor is like a switch, it either signals an entire information or none at all.

2.1 The Initial Logical Model

The initial logical model takes the form of implication rules. An example rule is the following (adapted from [6]):
$R1$: $(\neg\ pump1\_av \vee low\_wlvl)\ \wedge\ high\_temp\ \Rightarrow\ htemp\_unav\_circul$
It expresses that *the conjunction of the unavailability of circulation and a too high temperature of the fluid results in a detectable malfunction named htemp_unav_circul*.
The description of the variables used in the initial model follows: *lack_power* (lack of power supply), *low_temp* (low temperature threshold of the fluid), *high_temp* (high temperature threshold of the fluid), *pump1_en* (circulation pump engaged), *pump1_av* (circulation pump available), *low_wlvl* (low water level threshold), *high_wlvl* (high water level threshold), *pump2* (feed pump: *engaged* or *disengaged*), *fg1* (circulation floodgate: *open* or *closed*), *fg2* (feed floodgate: open or closed), *pump1_do_mp* (circulation pump disengagement order from MP), *pump1_do_rp* (circulation pump disengagement order from RP), *pump1_eo_mp* (circulation pump engagement order from MP), *pump1_eo_rp* (circulation pump engagement order from RP).
The other rules of the model have the same shape. We list only a few of them:

$R2$: $(\neg\ pump1\_av\ \wedge\ low\_wlvl)\ \Rightarrow\ unav\_circul$
There is unavailability of the circulation if the pump is unavailable and the water level is too low.
$R3$: $(pump1\_en\ \wedge\ low\_temp)\ \Rightarrow\ ex\_circul$
There is excess of water circulation if the circulation pump is engaged and the liquid temperature is too low.
$R4$: $(pump1\_eo\_mp\ \vee\ pump1\_eo\_rp)\ \wedge\ \neg\ lack\_power$
$\wedge\ \neg\ (pump1\_do\_mp\ \vee\ pump1\_do\_rp\ \vee\ fg1\_closed\ \vee low\_wlvl)$
$\Rightarrow\ \ pump1\_ego$
The conditions for global order of circulation pump engagement are: an engagement order from main or rescue panel, lack of electric power supply, no disengagement command from main or rescue panel, floodgate open and reservoir level not low.

3 A Reference Model for the Multi-facet Analysis

The proposed multi-facet analysis is based on a formal model which is a reference model for the forthcoming analysis tracks. The reference model should be consistent. Moreover it may be very abstract so as to enable some translations.

3.1 The Working Architecture for the Cooling System

The achievement of this step of the work may differ (due to their features) from one study to another. A preliminary study shows that the model for the cooling system can be completely derived from a control-oriented architecture due to the nature of the cooling system: it is a reactive system. Therefore we elaborate a suitable control-oriented architecture which emphasizes input and output variables. These variables will describe the states needed for the reference model: that is the *static part* of the model. The links between the input and output variables will determine the remaining *dynamic part* of the model. The resulting

architecture highlights *interacting* and *concurrent processes* which represent: *i)* sensors (*low_wlvl_sensor*, *high_wlvl_sensor*, *high_temp_sensor*, *low_temp_sensor*); *ii)* controlled processes (*floodgate1, floodgate2, pump1, pump2*); and *iii)* the cooling system. The architecure distinguishes between the controlled process and the software system. By combining the architecture and the initial model we can extract the variables describing input informations and output variables of the system. The input variables provide data from the sensors and orders given by the operator. The outputs of the system are the recommended operations and the states of the system components.

3.2 Building a Formal Reference Model

We build a model (of the entire system) to be used for both theorem proving and model checking aspects. It is made of a static part and a dynamic one.

Capturing the Static Part We characterize the cooling system by an abstraction $\mathcal{A} = \langle \mathcal{I}, \mathcal{S}, \mathcal{O}, \mathcal{R} \rangle$ where:

- $\mathcal{I}$ is a finite set of Boolean variables: the input variables,
- $\mathcal{S}$ is a finite set of Boolean variables: the state variables,
- $\mathcal{O}$ is a finite set of Boolean variables: the output variables and,
- $\mathcal{R}$ is a finite set of rules with the form: $condition \rightarrow variable$. An abstract form of a rule is a couple $(condition, variable)$. Moreover we have $\mathcal{R} \subseteq \mathcal{L}_{prop}(\mathcal{I}, \mathcal{S}) \times \mathcal{O}$.
  $\mathcal{L}_{prop}(\mathcal{I}, \mathcal{S})$ is the set of conditions (logical expressions) built using logical connectives and the variables in $\mathcal{I}$ and $\mathcal{S}$. In the following, the functions rhs and lhs denote respectively the right hand side and the left hand side of a rule. For example we have $\forall R_i \in \mathcal{R}.\mathsf{rhs}(R_i) \in \mathcal{O}$. Moreover we use the function vars to denote the set of variables appearing in the condition of a rule: $\mathsf{vars}(\mathsf{lhs}(R_i))$.

$\mathcal{R}$ captures (through the given rules) the relationship between $\mathcal{I}, \mathcal{S}$ and $\mathcal{O}$. Therefore, the logical rules express by their left hand sides the conditions of some malfunctions or the conditions under which some actions (order execution by the operator) are undertaken. The right hand sides of the rules are output variables.

Input Variables ($\mathcal{I}$): They describe the process. Mainly, they are informations coming from the all-or-none sensors; they are modelled with Boolean variables. The other inputs are the orders given by the operator. Logical propositions are used to describe the inputs of the process. Some input variables are: *lack_power*, *low_temp*, *high_temp*, *low_wlvl* and *high_wlvl*. If the operator triggers an action by an order, the associated variable is set to *true* otherwise it remains *false*. The orders given by the operator from the main panel and the rescue panel have the value *eo* (engagement order), *do* (disengagement order), *ego* (engagement general order), *dgo* (disengagement general order). The associated variables are *order_pump1_mp*, *order_pump1_rp* and *order_pump2_mp*.

State Variables ($\mathcal{S}$): They describe the control system and they are used as input and output informations. A state variable is associated to each malfunction. If the system detects a malfunction, the associated variable will be *true* otherwise the variable is *false*. In the same way, each engaged order has an associated state variable which indicates that the order is engaged or not. If the system engages an action by a command, the associated variable will be set. Therefore, the global state of the system is described by Boolean variables which represent the system component states. Besides this, *pump1* (resp. *pump2*) represents the circulation pump state (resp. the feed pump state). They can be engaged or not, disengaged or not, available or not. Circulation floodgate states and feed floodgate states are respectively modelled by the variable *fg1* and *fg2*. They are either closed or open.
Output Variables ($\mathcal{O}$): Output variables (including recommended operations) are calculated from input variables and state variables. If the system detects a malfunction the corresponding variable is set. $\mathcal{O}$ is partitionned by the sets $\mathcal{O}_m$ and $\mathcal{O}_o$.
$\mathcal{O}_m$ contains the output variables giving information about the state of the system: *unav_circul* (unavailable circulation), *htemp_unav_circul* (high temperature of the liquid and unavailable circulation), *ex_circul* (excess of circulation).
$\mathcal{O}_o$ contains the output variables giving orders: *pump1_dgo* (disengagement global order), *pump1_ego* (engagement global order), *pump1_el1* (locking of engagement of *pump1* by losing of power supply), *pump1_el2* (locking of engagement of *pump1* by losing of the liquid). The feed pump *pump2* is treated in the same way as *pump1*; hence the variables *pump2_dgo* (disengagement global order) and *pump2_ego* (engagement global order).

We have described on the basis of the system architecture, the static part of the reference model using appropriate type information, input, output and state variables. It is basically a state model from which we will derive isomorphic structures.
We discovered that this part of our work is related to the four-variable method suggested by Parnas to found rigorous software system development [7]. Therefore the four-variable model can be used as a more general reference model. The current experiment will prepare this purpose.

The Dynamic Part of the Model The cooling system scrutinizes current state and inputs, detects malfunctions, and helps to solve these malfunctions by interacting with the operator. This implies three main steps. The first consists in **reading the inputs** and events from the environment. Second, the logical rules are used to **detect malfunctions** and finally recommended operations are computed in order to **solve the malfunctions**. These three steps of the dynamic part will be detailed according to each analysis tool. For example concurrent asynchronous processes are convenient for Spin whereas operation modelled as generalized substitutions are used in B.

4 Safety Facet Study within B

We show in this section how the specific model for safety facet is systematically derived and used for the analysis.

4.1 Expressing the Reference Model within the B-method

The general principle here is to find a target structure corresponding to the reference model; the use of the type systems is a practical solution. The B method follows a state-oriented approach. Therefore, the static part of the B specification is described by an invariant (a predicate) on the variables of the system. The invariant describes the (good) state space of the specified system. It gives the types of the variables as well as the general properties of the system. We use the reference model variables to express the invariant. The state of the system is described by Boolean variables indicating the (sub)state of the system components. The B type system permits a straightforward translation of the reference model. We have an isomorphic model. The specification unit of the B method is an abstract state machine; it has a static part made of the variables and an invariant and, a dynamic part made of several operations. Here we build a B abstract machine (named *Process*) that describes the state space of the cooling system and that allows us to act on the states according to the input variables.

4.2 Specifying the Dynamic Part of the Model

The cooling system uses the rules to detect bad states and to indicate the orders to correct the bad states. The rules whose right hand side is an output variable, member of $\mathcal{O}_m$, are used to detect malfunction. Those whose right hand side is an output variable member of $\mathcal{O}_o$ are used to give orders. The rules are exploited to detect malfunctions and to solve them (by computing orders which are given to the operator).

Detecting Malfunctions The following is the transformation rule used to derive systematically B operations from the initial rules. The resulting operations are used to modify output variables.
Each logical rule with the form: $malfunction\_condition \Rightarrow malfunc\_name$ is systematically transformed into a B operation with the following shape:
set_malfunc_name = $malfunc\_name$:= **bool** ($malfunction\_condition$)
The B notation *var* := **bool***(condition)* stands for the classical conditional.
Formally, the **derivation principle** is a simple rewriting of the rule r_i defined as: $r_i \in \{r \in \mathcal{R} | \mathsf{rhs}(r) \in \mathcal{O}_m\}$

Therefore, for each output variable (called *malfunc_name*) appearing on the right hand side of a rule identifying a malfunction, we have a corresponding B operation (called **set_malfunc_name**).

Solving Detected Malfunctions It turns to be the calculation of output (orders for the operator) or state variables according to the current state variables and the input variables. We characterize each state variable of our system by a condition, called an *occurrence condition* and denoted by a function occur_cond. Besides, an identified malfunction rule has a *triggering condition* on its left hand side and a malfunction name on its right hand side:

$(trigger\_condition, malfunc\_name)$. To derive the B operation that corresponds to a rule to solve a malfunction, its triggering condition is examined. To solve the malfunction consists in modifying the process in such a way that this condition becomes false (the operator achieves an appropriate reaction to the system orders); then if the left hand side is false the malfunction has disappeared (the right hand side should be false).

From this analytical point of view, several rules are interrelated: the right hand side of the ones is used in the left hand side of the others. We derive systematically the B operations we need to complement the dynamic part of our model from these rules. For each rule with a right hand side $malfunc\_name$, we have a B operation named $solve\_malfunc\_name$ which has a body derived as explained in the following.

The **derivation principle** is as follows. First we compute for each rule R_j the set $\mathcal{R}_{R_j} = \{(sv, \mathsf{occur\_cond}(sv)) | sv \in \mathsf{vars}(\mathsf{lhs}(R_j))\}$. This associates to each state variable (sv) appearing in the triggering condition of a rule R_j, its occurrence condition. We define the procedure $RuleToB\_m$ which transforms a couple $(sv, occur\_cond)$ into `sv := bool(occur_cond)`; the latter has the form of a B substitution.
Applying this syntactic transformation (using $RuleToB\_m$) to the couples from $\mathcal{R}_{R_j}$ we get the set of substitutions in the body of the B operation which corresponds to the malfunction rule (the same for orders). Formally we obtained the substitutions of the B operations with: $\bigcup_{R_{vcond} \in \mathcal{R}_{r_o}} RuleToB\_m(R_{vcond})$
where $\mathcal{R}_{r_o} = \bigcup_{R_i \, \in \, \mathcal{R} | \mathsf{rhs}(R_i) \, \in \, \mathcal{O}_m} \{(sv, \mathsf{occur\_cond}(sv)) | sv \in \mathsf{vars}(\mathsf{lhs}(R_i))\}$.
These substitutions are then combined using the parallel operator $\|$. An example of a B operation derived by applying this derivation principle is as follows:

solve_low_wlvl = BEGIN
$pump2\_eo\_mp \; := \; bool(low\_wlwl = TRUE \; \vee \; pump2\_eo\_mp = TRUE) \; ||$
$fg2 \; := \; bool(low\_wlvl = FALSE \; \wedge \; fg2 = TRUE)$
END

Now we have two kinds of B operations derived from the reference logical model. We complement the static part of the *Process* abstract machine with a dynamic part made of all the operations. We add an INITIALISATION part where all the system variables are initialised in such a way that the system starts with a correct state.

Reading Inputs Due to the interactive nature of the process, the inputs cannot be systematically derived. Therefore, in this step, we introduce some specific operations to simulate the inputs from sensors and the inputs from the operator.

4.3 Verification of the Model: Consistency and Safety

Consistency and safety correctness preservation are handled conjointly through the invariant which gathers together all the properties of the system. Some safety properties are necessary so that the system works correctly. For example

– we cannot have sensors *low_wlvl* and *high_wlvl* detecting simultaneously a low and a high level of water in the reservoir:
$(low\_wlvl = FALSE \lor high\_wlvl = FALSE)$
$\land$ $(low\_temp = FALSE \lor high\_temp = FALSE)$
– the system should not order simultaneously an engagement and a disengagement of a pump:
$\neg$ $((pump1\_do\_mp = TRUE \lor pump1\_do\_rp = TRUE) \land$
$(pump1\_eo\_mp = TRUE \lor pump1\_eo\_rp = TRUE))$
$\land$ $(pump1\_dgo = FALSE \lor pump1\_ego = FALSE)$
$\land$ $(pump2\_dgo = FALSE \lor pump2\_ego = FALSE)$

The invariant part of the abstract machine is increased with these safety properties. Now every thing is in place for tool assistance. We use the tool Atelier B [8] to check the machines. The tool generates proof obligations which are the theorems to be proven. For the consistency one has to prove the following theorems: *i)* the initialization of the machines establishes the invariant and *ii)* each operation called under its precondition preserves the invariant. Since the invariant already contains safety properties, proving the preceding proof obligations guarantees safety.

Verification Result We check all the B machines and their implementations using the Atelier B. We have an amount of 239 proof obligations; they are all discharged. Two conclusions can be made: first, the machines are consistent (the model is consistent). Second, the correction (safety) of the system is established. Additionally, as far as refinement, code generation and simulation of the cooling system are concerned, we have refined the machine *Process* and elaborate a machine specifying the cooling system. In this latter machine we define a main operation as a supervision scenario. The scenario consists in doing a cyclic procedure (*working cycle*) where we have the previous three main steps.

5 Liveness Facet Study within Spin

The Spin (Simple Promela Interpretor) tool [2,3] is a model checker and more generally a verification tool that supports the design and verification of asynchronous processes. Promela is the input language of Spin. A Promela model is a CSP-like description of communicating asynchronous processes. At this stage of the work, we consider the two main parts of a Promela model: a static part made of state variable declarations and a dynamic part made of several asynchronous processes communicating via channels.

Expressing the Reference Model in Promela Using the type informations of the reference model, the translation of the static part into Promela is straightforward. Promela provides a sufficient type system for the translation. In Promela, *process types* are used to describe processes. A *process type* is the description of a behaviour using statements (arithmetic and logical expressions, sequences, repetitions, conditionals, guards, etc). Therefore we derive the dynamic part of the model by transforming the logical rules into process types. In the same way

as in the Section 4, we define for each involved step (Detecting Malfunctions, Solving Detected Malfunctions), the procedures to compute systematically the main part of the Promela model from the rules of the reference model. This is then complemented by processes which simulate the inputs from the sensors and the operator. Input informations are passed through channels between processes. After these steps, we have a Promela model made of the static part plus some asynchronous processes. Every thing is now in place for using the Spin tool.

Verification of the Model: Liveness The final model of the cooling system in Promela is a cyclic process (an iterative structure in Promela). Each cycle has the three steps explained above. The remaining steps of the approach are about the introduction of liveness properties and their validation on the model.
The Promela language offers some constructs to deal with property verification: *assertions, progress labels, acceptance labels* and *never claims* [2]. We use *never claims* in our experiment to capture the properties of the system. The `never claims` mechanism is a very expressive construct used to detect undesirable or illegal behaviours. Liveness properties are first specified using LTL formula and then translated into *never claims* mechanism with the Spin translator. The Spin translator gives the Büchi automata corresponding to the LTL formula [3]. In general we have to prove that if a malfunction appears then it will be solved. Some of these properties follow. P_1: *if a too high temperature of the fluid is indicated then the circulation pump will be activated*; P_2: *if a too low water level is indicated then the feed pump will be engaged*.
The *never claims* associated to these properties (their negated form) are joined to the validation model. Indeed we specify the desired properties and their negated forms are used as errors (undesirable states).

Validation Result The validation consists in ensuring that the model satisfies certain correctness properties. Given the Promela model with the associated properties described by `never` claims, we verify the correctness of the model using Spin. Spin explores the state space of the system, if an error occurs then it stops the exploration. Since *never claims* are used to detect undesirable states, it is required for a valid model to have zero error after a full state space search. Here is a brief report on the analysis of the P_1 property. In the case of full state search, there were 4453 generated system states which are stored (but 58228 states are matched); the longest non-cyclic execution sequence depth is 175, and no error is detected. 3.054 Megabytes are used (Spin uses a compressed form). In the current analysis experiment the maximum of stored states is 17961 (for a total of 787048 matched states); the non-cyclic execution depth in this case was 181 and 2.586 Megabytes are used. It arises from this experiment that the study is easily tractable with several small LTL properties (two or three modal operators).

6 Conclusions and Future Work

We presented an approach to tackle the multi-facet analysis of software systems using two tools (B and Spin) suitable to the facets of the system under anal-

ysis. The approach is illustrated with a case study where we considered safety and liveness facets. First we build a state model of the system, then we derive specific models using isomorphic structures. More generally the approach may help to tackle a system study where one needs various tools. Indeed, this leads to concentrate oneself on a single reference model plus some systematic transformation rules instead of moving through several starting models; which is error-prone. The proposed approach may be beneficially employed for the mechanization of integrated approaches [9]. In the way it is presented here the approach is a groundwork of an engineering framework to accompany the mechanization of formal methods. It is possible with reasonable efforts to help the users according to the characteristics of their systems to choose one reference model or another provided that their transformations into the input formats of various tools are known. Logic models, state models (for static aspects), labelled transitions systems, symbolic transition systems [10] (for dynamic aspects) are good candidates for such reference models with established transformation guides into specific tool formats. We are currently working on these aspects. The SAL tool [11,9] has similar goals. In addition, the generalization of the approach to various logics may favor interaction between tools and between reasoning systems.

Acknowledgement. *Thanks to the anonymous referees for their valuable comments.*

References

1. Abrial, J.R.: The B Book. Cambridge University Press (1996)
2. Holzmann, G.J.: Design and Validation of Computer Protocols. Prentice Hall, Englewoods, Cliffs (1991)
3. Holzmann, G.J.: The Spin Model Checker. IEEE Transactions on Software Engineering **23** (1997) 279–295
4. Shankar, N.: Combining Theorem Proving and Model Checking through Symbolic Analysis. In: Proc. of CONCUR'00. Volume 1877 of LNCS., Springer-Verlag (2000) 1–16
5. McMillan, K.L.: Verification of Infinite State Systems by Compositional Model Checking. In Pierre, L., Kopf, T., eds.: Proc. of Correct Hardware Design and Verification Methods. Volume 1703 of LNCS., Springer-Verlag (1999) 219–233
6. Gondran, M., Héry, J.F., Laleuf, J.C.: Logique et modélisation. Coll. DER - EDF. Eyrolles, ISSN 0399-4198 (1995)
7. Parnas, D.L., Madey, J.: Functional Documents for Computer Systems. Science of Computer Programming **25** (1995) 41–61
8. ClearSy: Atelier B V3.6. (Steria, Aix-en-Provence, France)
9. Attiogbé, C.: Mechanization of an Integrated Approach: Shallow Embedding into SAL/PVS. In: Proc. of ICFEM'02. Volume 2495 of LNCS., Springer-Verlag (2002) 120–131
10. Henzinger, T.A., Majumdar, R.: A Classification of Symbolic Transition Systems. In: Proc. of STACS 2000. Volume 1770 of LNCS., Springer-Verlag (2000) 13–34
11. Bensalem, S., Ganesh, V., Lakhnech, Y., Muñoz, C., Owre, S., Rueß, H., Rushby, J., Rusu, V., Saïdi, H., Shankar, N., Singerman, E., Tiwari, A.: An Overview of SAL. In Holloway, C.M., ed.: Proc. of the Fifth NASA LFM Workshop (LFM'2000), Vancouver (2000) 187–196