Ersch. in: MBEES 2013 : Tagungsband des Dagstuhl-Workshops ; Modellbasierte Entwicklung eingebetteter Systeme IX; Model-Based Development of Embedded Systems; 24.04.2013 - 26.04.2013 / Holger Giese ... - München : fortiss, 2013. - S. 60-69

Model-Based Quantitative Safety Analysis of Matlab Simulink / Stateflow Models

Adrian Beer, Todor Georgiev, Florian Leitner-Fischer and Stefan Leue

{adrian.beer, todor.georgiev, florian.leitner, stefan.leue}@uni-konstanz.de

Abstract: In this paper we report on work in progress to extend the QuantUM approach to support the quantitative property analysis of Matlab Simulink / Stateflow models. We propose a translation of Simulink / Stateflow models to CTMCs which can be analyzed using the PRISM model checker inside the QuantUM tool. We also illustrate how the information needed to perform probabilistic analysis of dependability properties can be specified at the level of the Simulink / Stateflow model. We demonstrate the applicability of our approach using a case study taken from the MathWorks examples library.

1 Introduction

The usage of model based engineering methods and tools is becoming state of the art in the development of safety-critical systems and software. The main reason for this trend is that model based engineering promises to facilitate dealing with the steadily increasing complexity of systems. In addition, safety standards like ISO 26262 [Int11] for the automotive domain or DO-178C [sta12] for the avionics domain recommend to use semi-formal or formal model based engineering methods in the design process. A prominent example of such a model based engineering tool is the Matlab Simulink / Stateflow tool-set of The MathWorks[1] which is widely used to design systems in the automotive and avionics domains.

ISO 26262 and DO-178C require that the safety of the system is analyzed using methods like, for instance, Failure Mode and Effects Analysis (FMEA) [Int91] or Fault Tree Analysis (FTA) [U.S81]. This gives rise to the question how safety and systems engineers can be supported in carrying out these analyses. The idea we pursue with the development or the QuantUM method and tool is that a) we support these techniques by automated formal analysis techniques such as model checking, and b) we embed these techniques into a model based engineering process which, as we argue above, is increasingly often the standard development practice in systems engineering.

In recent work [LFL11a, LFL11b, LFL12] we have provided such an embedding of formal dependability property analysis in a model based engineering process, in particular by providing an embedding for UML [uml11] and SysML [Sys10] based engineering processes in the QuantUM method and tool set. QuantUM is a framework which allows for the quantitative analysis of dependability properties for UML and SysML models [LFL11b]. In QuantUM, UML / SysML models edited using standard industrial CASE tools are automatically translated into the input language of the probabilistic model checker PRISM [HKNP06]. All information needed for the analysis of the model is specified on the level of the UML / SysML model. We thereby follow two principles frequently encountered in systems engineering practice: a) the model should contain both the normal and the anticipated failure

[1] http://www.mathworks.de/products/matlab/

behavior, and b) there should be a seperation between the modeling of normal and fault behavior, following the system engineering principle of seperation of concerns. In QuantUM, the information necessary for the above described analysis is specified by applying Stereotypes to the normal behavior UML / SysML model using specific QuantUM UML / SysML profiles. These stereotypes encapsulate information about fault modes and fault probabilities that will later be used to perform the formal dependability analysis. QuantUM thus separates normal behavior description from fault behavior specification. The QuantUM framework can then be used to answer dependability related questions such as "does the system satisfy its requirements 99% of the time?"

In this paper we report on work in progress to extend the QuantUM approach to support Matlab Simulink / Stateflow models. The envisioned QuantUM tool chain is depicted in Figure 1. The main step towards the goal of providing the same integrated analysis for Matlab Simulink / Stateflow as QuantUM already provides for UML and SysML is the translation of the Simulink / Stateflow models to the analysis model in the PRISM language, which is the focus of this paper. In particular, we focus on the treatment of the Stateflow part of Simulink which is the behavioral model of the language. Since Stateflow does not permit the use of stereotypes, a futher contribution of this paper is to illustrate how fault behavior information can be incorporated into Stateflow models while maintaining a separation between normal and fault behavior modeling.

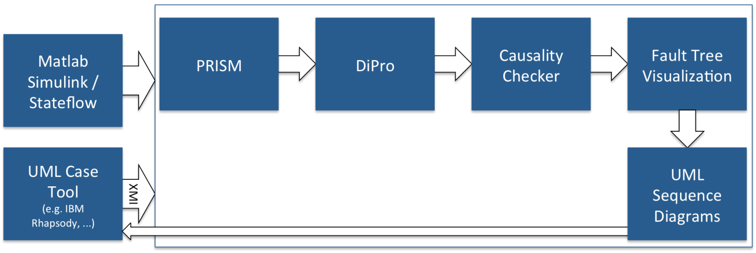


Figure 1: The QuantUM tool chain.

The remainder of this paper is structured as follows: The translation of Matlab Simulink / Statflow models to the analysis model used within QuantUM is discussed in Section 2. The applicability of the proposed approach is demonstrated using a cases study in Section 4. We discuss related work in Section 5 and conclude in Section 6.

2 From Simulink / Stateflow to CTMCs

Our semantics definition for the translation of Simulink / Stateflow models is based on work by Tiwari presented in [Tiw02]. Tiwari describes the semantics of transforming a subset of the Stateflow language into transition systems. The transition systems can be used to perform functional model checking with a functional model checker, e.g., Spin [Hol03]. In an intermediate step the notion of communicating pushdown automata is used.

Our aim is to enable a quantitative analysis of Simulink / Stateflow models. For this purpose we translate the Stateflow models into Continuous Time Markov Chains (CTMCs) [AKVR96]. CTMCs can be efficiently analyzed using the probabilistic model checker PRISM [HKNP06]. The failure rates of the different components have to be provided by the engineer and can be directly inserted in the Matlab environment, as later described in Section 4. We will focus on a translation of the Stateflow part of Simulink, since continuous behavior described by

Simulink blocks is not supported by the analyses offered by PRISM. Nevertheless, we want to cover an abstract behavior of the Simulink blocks so that the overall functionality can be analyzed. Therefore, we will propose a form of abstraction from the continuous behavior of Simulink blocks which, for now, has to be manually determined by the user.

Communicating Pushdown Systems. The notion of a communicating pushdown system (CPS) is used in an intermediate step of the translation to provide a formal basis for Stateflow charts [Tiw02]. They are in particular used to represent the hierarchical nature of the Stateflow state machines. A CPS consists of a set of pushdown automata (PDA). PDAs are constructed from a set of states, a stack, an input alphabet, a stack alphabet, and an initial state. A transition relation is used to change the current state based on the current input symbol or the symbol on the top of the stack. When a transition is executed it can either put a new symbol on the stack, pop the top symbol or leave the stack unchanged. We will use a stochastic version of the CPS where the transition relation is replaced by a transition rate function, which additionally carries information about the transition rate.

Mapping Stateflow to CTMCs. In the Simulink / Stateflow modeling language the graphical position of the drawn components influences the semantic interpretation of the execution order, see for instance the ”twelve-o'clock-rule” of the Stateflow semantics. This means that small variations in the graphical layout can have significant impact on the semantic interpretation of a chart. We believe that this feature leads to ambiguous interpretations of a Stateflow diagram. Therefore, we will not rely in our interpretation on the graphical layout but use a concurrent semantic interpretation instead, as also proposed in [PMH$^+$02]. A Stateflow model is translated into a communicating pushdown system, i.e., the different AND-states and components are translated into a set of stochastic pushdown automata (PDA). Afterwards, each PDA is further translated into a CTMC. The CTMCs are then associated with PRISM modules representing the different components. PRISM uses the standard semantics for communicating sequential processes (CSP) to perform a parallel composition of the different modules [Hoa78].

3 Translation Challenges

Continuous Behavior. In Simulink / Stateflow signals can be used to express continuous behavior of Simulink blocks. In [Tiw02] some suggestions are presented to deal with such continuous behavior. For now, we use the most simple way of treating the continuous behavior: We abstract those Simulink blocks which incorporate continuous signals into discrete decision blocks. This means that the engineer has to provide some boundary behavior of these blocks and the related continuous signals. For example, if a block generates some continuous output, like pressure in a hydraulic system, the output is abstracted to a binary decision: either high pressure, or low pressure. The implementation of the automatic symbolic detection of these boundaries, as proposed in [Tiw02], is left for future work.

The PRISM Language. The translation of Simulink / Stateflow models into CTMCs gives rise to several challenging issues. Most of them are due to limitations of the PRISM input language itself.

Asynchronous communication is not supported in PRISM, even though it is possible to implement modules in PRISM which act as bounded channels (see [FG06]). An easier way of mimicking asynchronous communication in PRISM is to use local variables which are written at some synchronized points and then read asynchronously. This kind of pseudo asynchronous communication can only be used when one kind of message can only be sent once and is surely being processed until the next sending of the message. We will use this kind of message handling in our case study. This is justified since we only have to handle messages when a failure occurs, and the message then occurs only once.

The other challenge arises when the Stateflow chart is built in a hierarchical manner. The definition of the Stateflow semantics says that if the parent state of a Stateflow chart can be left, all descendant states have to be left as well and their exit action has to be executed. The leaving of the descendant states has to be accomplished by additional guards in the PRISM language. The exit action of the descendant states can only be taken if we generate an extra transition in the PRISM code for every descendant. This construction leads to an exponential blow-up of the transitions with the depth of the hierarchy during generation of the PRISM code.

Simulink / Stateflow Limitations. The notion of Stereotypes that we use to specify the fault behavior for UML and SysML models is not available in Simulink / Stateflow. This makes it more difficult to incorporate fault behavior into the model while separating it from the normal system behavior. We also need to specify what we refer to as the failure pattern, which is a description of the behavior of a failed component. In UML it is very easy to add another state machine to the model in order to represent the failure behavior and to associate this machine with a component. In Simulink / Stateflow it is impossible to associate more than one independent Stateflow diagram directly with a component. In our first prototypical implementation we add the failure pattern directly to the normal behavior model, which violates the QuantUM goal of a separation of concerns, as argued earlier. A possible solution to this problem is to add for each failure mode an additional Simulink block with a Stateflow chart representing the failure pattern to the normal behvior model. This chart would have one failure transition that is synchronized with the normal behavior. It would get triggered when the failure occurs. This is not as clear a separation than the stereotype based solution for UML/SysML, but still it allows for some encapsulation of the fault behavior.

4 Case Study

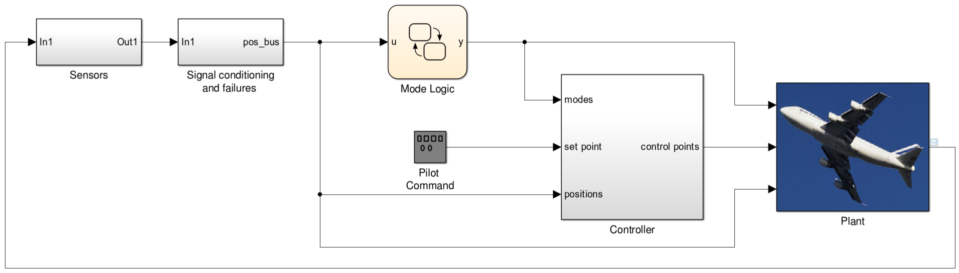


Figure 2: An Overview of the elevator control system. [MG04]

We illustrate our approach using an Aircraft Elevator Control System. This example is taken from the examples repository of the Matlab Simulink / Stateflow framework and was introduced in [MG04]. An overview of the system is given in Figure 2. Usually, an aircraft has two elevators attached to the horizontal tails. The system consists of two independent hydraulic actuators per elevator. The actuators are driven by three separate hydraulic circuits. The system is controlled by two primary flight control units (PFCU) and two control modules per actuator. One control module operates according to a full range law and the other one works according to a reduced range law. The reduced range control is only used in emergency cases when the first control module has failed. In the case study we limit the system to a subset of the original functionality to reduce the complexity of the model. Each PFCU will have only one control law, and controls only set one of the actuators. One actuator per elevator runs the full range law and the other one runs the reduced range law. The full range actuators both have a dedicated hydraulic circuit, while the reduced range actuators share one circuit.

In Figure 3 an example Stateflow chart for a full range actuator is shown. In the default setting the full range actuators are on, and the reduced actuators are on standby. If a fault is detected in the active actuators or in one of the hydraulic circuits that are connected to them, the system shall respond by turning the full range actuators off and the standby actuators on. The FMEA conducted for the Elevator System in [MG04] showed that for a single failure in the system, for example either a position failure of an actuator or a hydraulic pressure failure, there is always one active actuator per side. For the occurrence of a combination of failures there is a certain probability of leaving the system in an active state, but the authors of [MG04] are not giving any actual probability values for the system. We would hence like to check our variant of the model for the property “the probability of losing both actuators on one side, either right or left” using QuantUM. This case would lead to an uncontrollable situation for the pilot and can hence be considered a hazard.

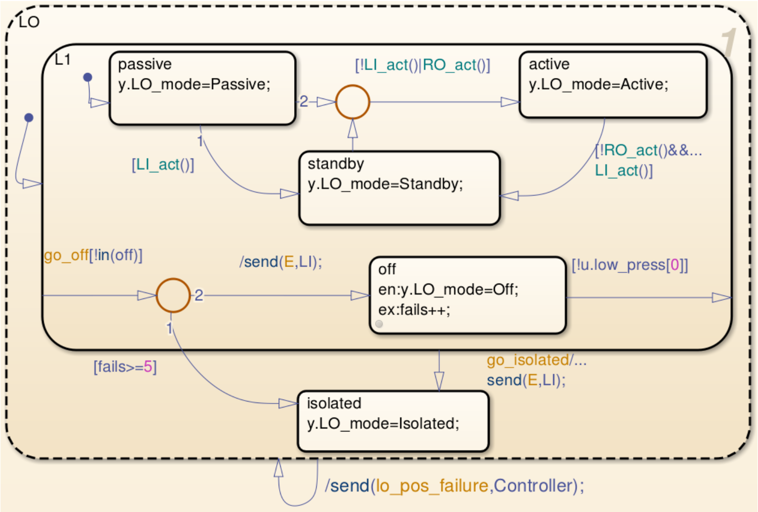
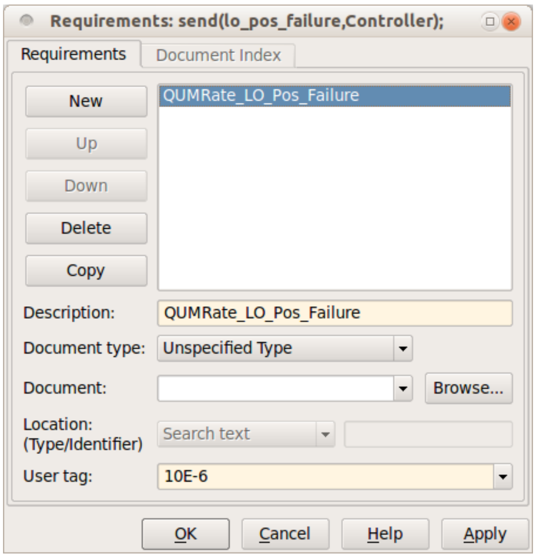


Figure 3: The Stateflow chart for the left full range actuator (LO)

In Figure 4 the requirements dialog of Simulink / Stateflow is shown. In this dialog the user can enter additional requirements for different items in the charts. Here we show the requirements dialog for a failure transition of the full range actuator which is tagged with a failure rate of 10^{-6}. We are using guessed values for the rates of the different components, since we don’t have access to real values. For the actuators we are using failure rates in the order of 10^{-6} and for failures in the pressure cycle we are using rates in the order of 10^{-4}. We maintain that the guessed values are in the order of magnitude of realistic values.


Requirements: send(lo_pos_failure,Controller);

QUMRate_LO_Pos_Failure

Description: QUMRate_LO_Pos_Failure

Document type: Unspecified Type

Document:

Location: (Type/Identifier) Search text

User tag: 10E-6

Figure 4: The requirements dialog of MatLAB where the failure rates can be entered.

As discussed above, we need to abstract the continuous behavior introduced with the Simulink blocks. First, we abstract the continuous behavior of the hydraulics system. We introduce a new hydraulic system model which switches the modes of the three hydraulic systems probabilistically to either high/normal pressure or low pressure. Additionally, we abstract the sensor behavior for detecting faults in the actuators to a probabilistic decision. An actuator failure can be a faulty positioning of the elevators. The PFCU is adapted to this new behavior by switching over to a behavior based on the probabilistic failure conditions. As discussed above, we have modeled the failure modes and conditions directly inside the Simulink / Stateflow blocks.

Example Translation In Listing 1 the manually translated PRISM Code for the left full range actuator from Figure 3 is shown. The states are represented through the variable `states_LO`. The state active from the Stateflow chart, for example, is represented by `states_LO` having the value 5. The initialization procedure for this actuator starts at the initial state of the Stateflow chart LO (Line 11) and further the initial state of the sub Stateflow chart L1 (Line 12). The receiving of events is represented by the transition labels at the beginning of each transition in the code, for example, the `go_off` event which switches the actuator in the off mode is received in Line 28. The reception of the event results in the variable `LO_go_off` being set to true which enables the corresponding transition (Line 29). This is the type of asynchronous communication discussed in Section 3. In Line 41 the failure transition is displayed which has the failure rate for the positioning failure attached. Some additional mapping examples for the different elements in the Stateflow chart can be found in Table 1. These mappings will serve as a starting point for the definition of a comprehensive set of semantic mapping rules.

```
module LO

 states_LO: [0..9] init 0;
 LO_fails:[0..6] init 0;
 LO_go_off:bool init false;
 LO_active:bool init true;
 LO_pos_failure:bool init false;
 reactivate_LO:bool init false;

 // go into passive mode
 [LO_tr1] (states_LO=0)->(states_LO'=1);
 [LO_tr2] (states_LO=1)&(!LO_go_off)->(states_LO'=2);
 // go active or standby
 [LO_tr3] (states_LO=2)&(!reactivate_LO)&(LI_active=true)&(!LO_go_off)
    ->(states_LO'=3);
 [activate_LO] (states_LO=2)&(!LO_go_off)
    ->(states_LO'=4)&(LO_active'=true);
 [LO_tr5](states_LO=2)&(reactivate_LO)&(!LO_go_off)
    ->(states_LO'=4)&(reactivate_LO'=false);
 [LO_tr6](states_LO=4)&(LI_active=false|RO_active=true)&(!LO_go_off)
    ->(states_LO'=5)&(LO_active'=true);
 [LO_tr7](states_LO=5)&(RO_active=false&LI_active=true)&(!LO_go_off)
    ->(states_LO'=3)&(LO_active'=false);
 [LO_tr8](states_LO=3)&(!LO_go_off)
    ->(states_LO'=4);
 // Fault detection
 // if LO isn't allready off or isolated then we turn it off
 [LO_event_go_off](!states_LO=9)&(!states_LO=8)->(LO_go_off'=true);
 [LO_tr9](children_states_LO_L1)&(LO_go_off)
    ->(states_LO'=6)&(LO_active'=false);
 [LO_tr10](states_LO=6)&(LO_fails>=5)
    ->(states_LO'=9);
 [LO_tr11](states_LO=6)&(LO_fails<5)
    ->(states_LO'=8)&(LO_fails'=LO_fails+1)
      &(LO_go_off'=false)&(reactivate_LO'=true);
 // if this wasn't LO's fault turn it passive again
 [LO_tr12](states_LO=8)&(high_press0=true)->(states_LO'=1);
 // if there was a major failure turn LO off completely
 [LO_event_gois](!states_LO=9)->(states_LO'=9)&(LO_active'=false);
 //Failure Model
 [LO_pos_failure](!states_LO=9)->10E-6:(LO_pos_failure'=true);

endmodule
```

Listing 1: The generated PRISM code for the full range actuator shown in Figure 3

Results. All calculations of the probabilities with PRISM were done on an Intel Core i7 with 3.33GHz and 24GB of RAM. We verified the properties over a mission time of 1000 hours. As demanded by the specification the model checking result for single failures is always a probability of 0.0 since, as mentioned above, a single failure always leaves the system in an active state. The time needed by PRISM to calculate the probability of all single failures was less than one second. The combination of a positioning failure in either both full range or both limited range actuators also gives a failure probability of 0.0, since the control is directly transfered to the other actuators, respectively.

For a combination of a positioning failure for one side of the elevator system, for example the left full range and the left limited range actuators, the probability of a total loss was 9.86^{-5}. The calculation of the above probability was done by PRISM in less than one second.

Stateflow:	PRISM Reference:
AND state LO	module LO (Line 1)
initial state of LO	states_LO=0 (Line 3)
sub Stateflow chart L1 (normal behavior)	states_LO > 0 AND states_LO ≥ 5 (Line 14-25)
The asynchronous reception of the go_off event	if the event was received the variable LO_go_off is set to true (Line 5 + 28)
activation of the go_off transition	deactivation of the transitions (Line 12-25) via (!LO_go_off) + activation of the transition (Line 29) via (LO_go_off)
failure handling	states_LO > 5 AND states_LO ≥ 8 (Line 28-37)
go_isolated event and isolated state	states_LO=9 (Line 39)
the positioning failure transition + sending of LO_pos_failure to Controller	LO_pos_failure synchronization label with a rate of 10^{-6} (Line 41)
Guard [!RO_act() && LI_act()] for the transition from active to standby	(RO_active=false&LI_active=true) (Line 22)

Table 1: Examples for the mapping of the Stateflow chart in Figure 3 to the PRISM code in Listing 1

A more complex combination of failures, like a combination of the hydraulic cycles for the full and the limited range actuators, resulted in a considerably larger reachable part of the CTMC with 7.588.856 states and 46.377.640 transitions. The calculation time grew up to 47.44 minutes. The resulting probability was 4.756^{-11}. The size of the system can be explained by the more complex failure handling system for a failure in the hydraulic pressure and the resulting exponential blow up of the state space due to the concurrent execution of the components.

5 Related Work

There is a significant body of work on a formal treatment of Simulink and Stateflow available in the literature. Simulink possesses a build-in verification tool called Simulink Design Verifier, which is a model checker based on SAT solving technique [ABCH02]. It does not support concurrency, non-determinism and detection of deadlocks, as discussed in [LFL08]. There are works describing the translation into various qualitative model checking languages, including NuSMV, Lustre, SAL and Promela/SPIN [BKB99, MBR06, SSC$^+$04, TSCC05, SCBR01, PMH$^+$02, LFL08]. In order to be able to treat continuous behavior the translation of Simulink / Stateflow models into Hybrid Automata has been proposed in [SRKC00, HHWT97, ADE$^+$01]. However, this approach possesses very limited scalability.

In spite of the large amount of work describing various translations, there are only few discussions for a semantic formalization of Simulink / Stateflow. Hamon describes an operational semantics of Stateflow charts which uses SAL as an output language [Ham05, HR04]. In [BM06] both Simulink and Stateflow are formalized based on the mode automata design method. As mentioned earlier, a formal semantics of Simulink / Stateflow based on communicating pushdown automata is presented in [Tiw02]. To the best of our knowledge, there is no translation from Simulink / Stateflow to any probabilistic or stochastic model checking language. We are also not aware of any work extending Stateflow by probabilistic constructs.

6 Conclusion and Future Work

We have presented ongoing research on incorporating probabilistic, automated dependability analysis based on probabilistic model checking of Simulink / Stateflow models into the QuantUM method and toolset that we currently develop. We discussed issues in the semantics of Stateflow / Simulink that need to be addressed. We sketched a possible translation from Stateflow charts to PRISM code that awaits further formalization. We have illustrated the feasibility of this approach using a Simulink / Stateflow model of non-trivial complexity.

In future work we plan to fully automatize the translation mechanism and integrate it into our QuantUM framework. Furthermore, an automatic symbolic boundary detection for contiuous variables / flows is planed in order to minimize the effort needed from the user to conduct a complete analysis.

References

[ABCH02] Gunnar Andersson, Per Bjesse, Byron Cook, and Ziyad Hanna. A proof engine approach to solving combinational design automation problems. In *Design Automation Conference, 2002. Proceedings. 39th*, pages 725–730. IEEE, 2002.

[ADE$^+$01] Rajeev Alur, Thao Dang, Joel Esposito, Rafael Fierro, Yerang Hur, F Ivančić, Vijay Kumar, Insup Lee, Pradyumna Mishra, George Pappas, et al. Hierarchical hybrid modeling of embedded systems. In *Embedded Software*, pages 14–31. Springer, 2001.

[AKVR96] A. Aziz, K. Sanwal, V. Singhal, and R. K. Brayton. Verifying Continuous-Time Markov Chains. In *Proc. of CAV 1996*, volume 1102 of *LNCS*, pages 269–276. Springer, 1996.

[BKB99] Chonlawit Banphawatthanarak, Bruce H Krogh, and Ken Butts. Symbolic verification of executable control specifications. In *Computer Aided Control System Design, 1999. Proceedings of the 1999 IEEE International Symposium on*, pages 581–586. IEEE, 1999.

[BM06] Pontus Boström and Lionel Morel. *Mode-Automata in Simulink/Stateflow*. Citeseer, 2006.

[FG06] Ansgar Fehnker and Peng Gao. Formal verification and simulation for performance analysis for probabilistic broadcast protocols. In *Proceedings of the 5th international conference on Ad-Hoc, Mobile, and Wireless Networks*, ADHOC-NOW'06, pages 128–141, Berlin, Heidelberg, 2006. Springer-Verlag.

[Ham05] Grégoire Hamon. A denotational semantics for Stateflow. In *Proceedings of the 5th ACM international conference on Embedded software*, pages 164–172. ACM, 2005.

[HHWT97] Thomas A Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HyTech: A model checker for hybrid systems. *International Journal on Software Tools for Technology Transfer (STTT)*, 1(1):110–122, 1997.

[HKNP06] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A Tool for Automatic Verification of Probabilistic Systems. In *Proceedings of TACAS 2006*, volume 3966 of *LNCS*. Springer, 2006.

[Hoa78] Charles Antony Richard Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[Hol03] Gerhard J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addision–Wesley, 2003.

[HR04] Grégoire Hamon and John Rushby. An operational semantics for Stateflow. *Fundamental Approaches to Software Engineering*, pages 229–243, 2004.

[Int91] International Electrotechnical Commission. Analysis Techniques for System Reliability - Procedure for Failure Mode and Effects analysis (FMEA), IEC 60812, 1991.

[Int11] International Organization for Standardization. Road Vehicles – Functional Safety, ISO 26262, 2011.

[LFL08] F. Leitner-Fischer and S. Leue. Simulink Design Verifier vs. SPIN - A Comparative Case Study. In *Participant's Proceedings of FMICS 2008, ERCIM Working Group on Formal Methods for Industrial Critical Systems*, 2008.

[LFL11a] F. Leitner-Fischer and S. Leue. Quantitative Analysis of UML Models. In *Proceedings of Modellbasierte Entwicklung eingebetteter Systeme (MBEES 2011). Dagstuhl, Germany.*, 2011.

[LFL11b] Florian Leitner-Fischer and Stefan Leue. QuantUM: Quantitative Safety Analysis of UML Models. In *Proceedings Ninth Workshop on Quantitative Aspects of Programming Languages (QAPL 2011)*, volume 57 of *EPTCS*, pages 16–30, 2011.

[LFL12] Florian Leitner-Fischer and Stefan Leue. Towards Causality Checking for Complex System Models. In *Proceedings of Modellbasierte Entwicklung eingebetteter Systeme (MBEES 2012). Dagstuhl, Germany.*, 2012.

[MBR06] B Meenakshi, Abhishek Bhatnagar, and Sudeepa Roy. Tool for translating simulink models into input language of a model checker. *Formal Methods and Software Engineering*, pages 606–620, 2006.

[MG04] Pieter Mosterman and Jason Ghidella. Model reuse for the training of fault scenarios in aerospace. In *AIAA Modeling and Simulation Technologies Conference and Exhibit*, 2004.

[PMH$^+$02] Paula J Pingree, E Mikk, GJ Holzmann, MH Smith, and D Dams. Validation of mission critical software design and implementation using model checking [spacecraft]. In *Digital Avionics Systems Conference, 2002. Proceedings. The 21st*, volume 1, pages 6A4–1. IEEE, 2002.

[SCBR01] Steve Sims, Rance Cleaveland, Ken Butts, and Scott Ranville. Automated validation of software models. In *Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on*, pages 91–96. IEEE, 2001.

[SRKC00] B Izaias Silva, Keith Richeson, Bruce Krogh, and Alongkrit Chutinan. Modeling and verifying hybrid dynamic systems using CheckMate. In *Proceedings of 4th International Conference on Automation of Mixed Processes*, pages 323–328, 2000.

[SSC$^+$04] Norman Scaife, Christos Sofronis, Paul Caspi, Stavros Tripakis, and Florence Maraninchi. Defining and translating a safe subset of simulink/stateflow into lustre. In *Proceedings of the 4th ACM international conference on Embedded software*, pages 259–268. ACM, 2004.

[sta12] DO-178C/ED-12C: Software Considerations in Airborne Systems and Equipment Certification, 2012.

[Sys10] Systems Modelling Language, Specification 1.2, Jun. 2010.

[Tiw02] A. Tiwari. Formal semantics and analysis methods for Simulink Stateflow models. *Technical report, SRI International*, 2002.

[TSCC05] Stavros Tripakis, Christos Sofronis, Paul Caspi, and Adrian Curic. Translating discrete-time simulink to lustre. *ACM Transactions on Embedded Computing Systems (TECS)*, 4(4):779–818, 2005.

[uml11] Unified Modelling Language, Specification 2.4.1, August 2011.

[U.S81] U.S. Nuclear Regulatory Commission. *Fault Tree Handbook*, 1981. NUREG-0492.