Scade2Nu : A Tool for Verifying Safety Requirements of SCADE Models with Temporal Specifications

Jian Shi[1]
jian.shi@ntesec.ecnu.edu.cn

Jianqi Shi[1,2,*]
jqshi@sei.ecnu.edu.cn

Yanhong Huang[1,2]
yhhuang@sei.ecnu.edu.cn

Jiawen Xiong[1]
jiawen.xiong@ntesec.ecnu.edu.cn

Qing She[2]
qing.she@ntesec.ecnu.edu.cn

[1]National Trusted Embedded Software Engineering Technology Research Center
East China Normal University, Shanghai, China
[2]Hardware/software Co-Design Technology and Application Engineering
Research Center, Shanghai, China

Abstract

SCADE is both a language and a model-based software development environment that can develop systems in safety-critical fields. It is paramount for these systems to satisfy their safety requirements. Although SCADE can verify some properties by its Design Verifier (DV), it cannot verify the unbounded temporal or liveness properties due to its limitation. With the diversification of system application scenarios, these temporal properties also need to be verified to ensure the reliability of the system. In this paper, we concentrate on solving this problem by introducing Scade2Nu, a tool that can transform **SCADE** state machines in**to** **Nu**SMV input languages. Scade2Nu helps the designer to apply Linear-time Temporal Logic (LTL) and Computational Tree Logic (CTL) formulas as the requirements specifications of SCADE models. With the aid of the NuSMV model checker as well as its verification results, designers can explore more different kinds of properties to further reduce bugs at the requirements phase in the development cycle.

1 Introduction

Embedded software in fields like aerospace, rail transportation, industrial and nuclear energy often requires safety in analysis and high-security standards. SCADE (Safety Critical Application Development Environment) can design critical systems such as emergency braking systems, power and fuel management, automatic train operation and many other industrial applications. It has broad application in safety-critical fields also because of its compliance with safety standards such as DO-178B/C, IEC 61508 and EN 50128. Most of these standards define the guidelines that the verification of requirements for systems can minimize the risk of error introduction in the design phase and ensure the effectiveness of the left software development process.

Design Verifier[ADS+04] (DV) is the formal verification component for SCADE, and it expresses safety requirements with SCADE language and performs SAT-based model checking on SCADE models. However, as

[*] Corresponding Author

the number of SCADE users grows, the stricter, more diverse temporal properties arising from the safety-critical requirements will be for verification purposes. In formal verification, the popular alternative methods for specifying most of these properties are temporal logics, such as Linear Time Logic (LTL) or Computation Tree Logic (CTL). However, DV cannot adequately express general temporal specifications. In order to analyze the liveness property in industrial practise, Daskaya et al.[DHM11] have transformed the SCADE state-based model into UPPAAL. Therefore, the capability of verifying various temporal properties of SCADE models is an expected demand by SCADE users. If it would support temporal logics, unbounded temporal specifications could be expressed and checked as well so that more defects can be detected and eliminated earlier in the design phase in the development cycle. Therefore, we present Scade2Nu, a tool that can use temporal logics like LTL and CTL specifications to verify whether the safety-critical requirements of control systems designed by SCADE have been met.

In SCADE, engineers often use its state machine to design critical control systems. In the high-level, the structure of the SCADE state machine (SSM) is similar to that of the StateChart[Dor08][Har87], which means it has the characteristic of hierarchical state machine[AY01]. Taking it into consideration, Scade2Nu translates the SCADE state machine models into the input language of NuSMV[1]. The reasons why we choose NuSMV are ① the strict correctness of its model language, ② the ability to express hierarchical models and ③ the effective model checking algorithms. The translator framework we applied in Scade2Nu is based on the STP-approach presented by Clarke et al.[CH00]. They defined a modular translation rule through a temporal language ETL, which correctly describes the transformation from StateChart to SMV. Since the actions in SCADE models are more concrete so we refine the monitor-like mechanism in STP-approach as SCADE variables-monitor mechanism in the implementation of Scade2Nu so that it can accommodate for the SCADE situation.

In the following, we present an overview of Scade2Nu in Section 2. Section 3 will show the usage of Scade2Nu via a case study. Finally, the conclusion and future works will be presented in Section 4.

2 Overview of Scade2Nu

Scade2Nu brings the benefits of temporal verification technology to the SCADE requirements verification. The verification framework and architecture of Scade2Nu are illustrated in Fig.1. This process starts with a SCADE model and its safety requirements, then translates SCADE models into NuSMV programs. Scade2Nu has the following components that make it do the translation:

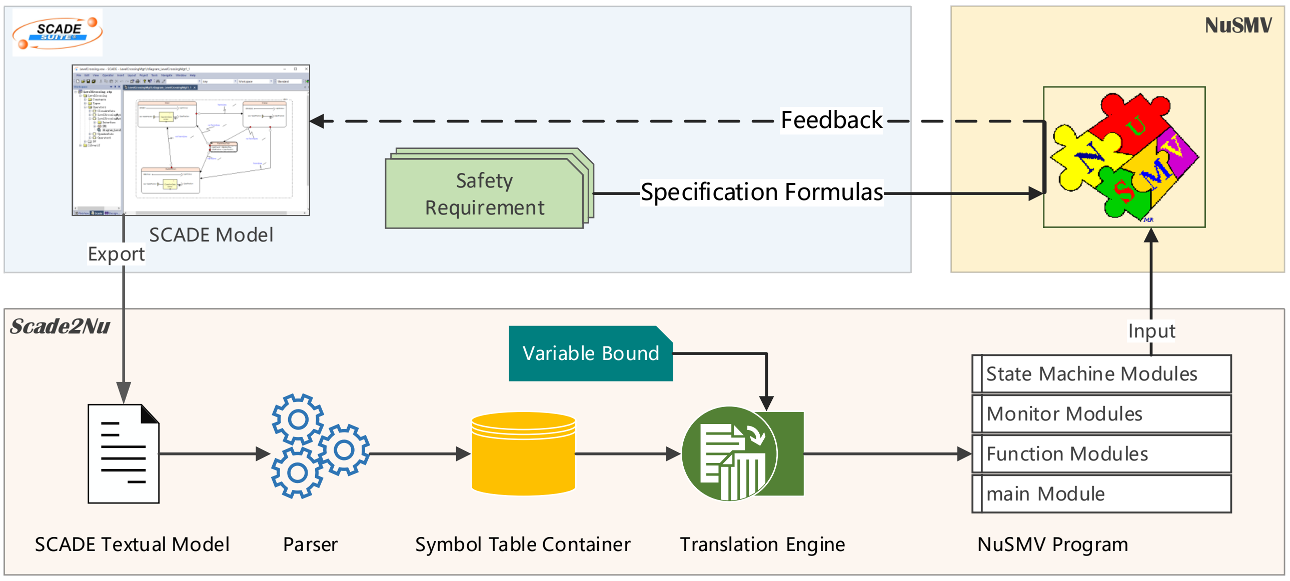


Figure 1: Verificaition Framework for the SCADE Model in Scade2Nu.

SCADE Textual Model Parser. The textual model is the SCADE model represented by the SCADE language. We rewrite a grammar file according to its reference manual[ET15] and parse the textual model using

[1]A symbolic model checker, see http://nusmv.fbk.eu/

ANTLR[2]. This parser generates the parse tree and provides the base interface for tree visiting.

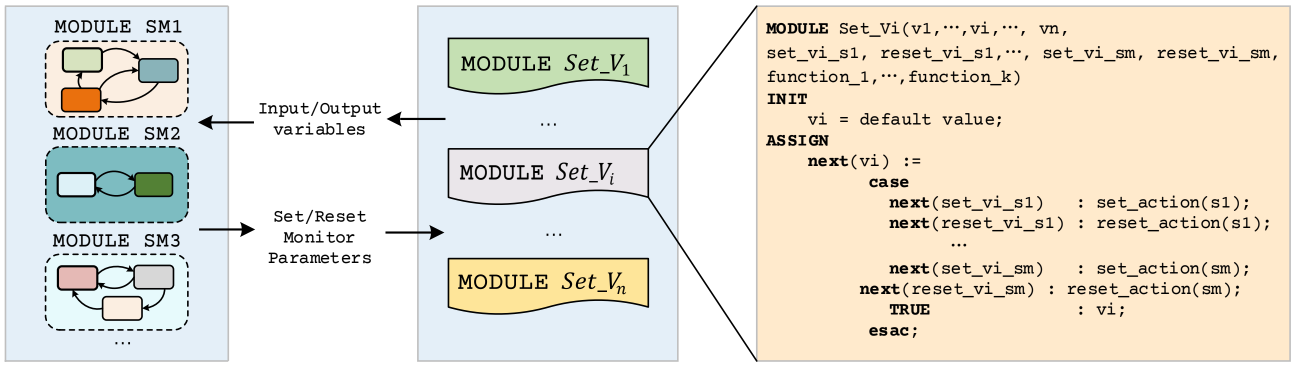
Symbol Table Container. This component accepts the parse tree, extracts its information and stores them into a symbol table. The symbol table is a data structure of the hierarchical SCADE State Machine (SSM). We define it with a tuple $\langle I, O, SM, Op, Fun \rangle$ where I and O are the set of input variables and output variables respectively. SM is a set $\{SM_1, ..., SM_n\}$ where each SM_i represents the structure of a sub-state machine. The basic operation blocks that SSM used are stored in Op, while Fun has all function nodes that the SSM used. The operations library that Scade2Nu supports includes basic operations, such as mathematical operations, comparison operations, logical operations, choice operations, and temporal operations like previous. And we concentrate on functions in data-flow forms. A function node has its input variables, output variables, and equations so that it can operate computations.

Scade2Nu implements many listeners and visitors using the ANTLR runtime API so that it can get all information about the SCADE state machine and pack it in an SSM symbol table. However, before we put an SSM into translation engine, we had better use abstraction techniques so that verification of NuSMV models can run faster. Though we know that NuSMV model checker is a symbolic model checker that can handle systems with large space size, we should also provide an option which bounds the ranges of some or all variables in Scade2Nu for engineers. The benefit of this simple abstraction is helping to further reduce the state space size.

Translation Engine. In this step, Scade2Nu puts the SSM instance into the translation engine and generates four parts of the target model based on translation rules : ① state machine modules `SMi`, ② monitor modules `Set_Var`, ③ `Function` modules, and ④ the `main` module. Translation rules that Scade2Nu implements are based on STP-approach. The detials of the rules and the mechanismcan can be found in our online resource[3]. These translation rules are classified into two aspects. The engine applies the rules about the hierarchical structure of state machines and refines the rules of a monitor-like mechanism.

Hierarchical Structure. Each sub-state machine in the SCADE models corresponds to one state machine module `SMi` in the NuSMV program. These modules are built in a hierarchical structure through two parameters `active` and `default` that introduced in STP-approach. The engine follows the translation rules to generate the hierarchical state machine modules.

SCADE Variable Monitor Mechanism. In STP-approach, the monitor-like mechanism means for every event or condition variable `var` there is a monitor module `Set_Var` which is a solo module to manipulate the variable `var` via monitor parameters. But the problem is that its full abstraction of conditions and events will ignore the concrete actions in SCADE states. In SCADE state machine, one state `s` may do specific operations on their output variables. Therefore, we refine the two forms of monitor parameters set_m and $reset_m$ in STP-approach to $set_{(var,s)}$ and $reset_{(var,s)}$.



```
MODULE Set_Vi(v1,⋯,vi,⋯, vn,
set_vi_s1, reset_vi_s1,⋯, set_vi_sm, reset_vi_sm,
function_1,⋯,function_k)
INIT
    vi = default value;
ASSIGN
    next(vi) :=
          case
            next(set_vi_s1)   : set_action(s1);
            next(reset_vi_s1) : reset_action(s1);
                 ⋯
            next(set_vi_sm)   : set_action(sm);
         next(reset_vi_sm) : reset_action(sm);
            TRUE              : vi;
          esac;
```

Figure 2: SCADE Variable-Monitor Mechanism.

Assume that $v_1, ..., v_i, ..., v_n$ are the variables of an SSM, $s_1, ..., s_m$ denote the states that have behaviors about v_i. Fig.2 shows what the monitor module for variable v_i looks like after translation. It declares the related monitor parameters $(re)set_{(var,s)}$. And the $(re)set\_action$ is the assignment expression for v_i that need to be performed in respective states. They are data-flow operations in states, and the engine translates these operations

[2]ANother Tool for Language Recognition, see https://www.antlr.org/
[3]Online appendix.

into corresponding expressions straightforwardly. Some of them apply other function nodes that user-defined. So, Scade2Nu can declare parameters $function_1, ..., function_k$ if actions need.

In the left of Fig.2 shows the SCADE variable monitor mechanism after we refine the monitor parameters. The variables may change the transition condition expressions so that they can make state machines do state transitions. Each sub-state machine module `SMi` reads these variables and operates the assignment of monitor parameters based on STP-approach. So it rewrites the values of monitor parameters so that these parameters can decide the values of output variables through the assignment in the monitor modules. This is how the monitor parameters manipulate the state machines indirectly.

`Function` modules are generated from Fun. Each function has its own variables and equations. Since function nodes are in data-flow forms, the translation of each equation is straightforward. In the final NuSMV model, there is only one `main` module that handles the top-level state machine. It instantiates its sub-state machines, monitor modules, and function modules, and generates the declarations of variables and monitor parameters.

3 Case Study

In this section, we illustrate the procedure to verify the safety-critical requirements of a cruise control system using Scade2Nu. The cruise control is a industrial system created by Esterel Technologies, which can control the speed of a car automatically. The driver sets the pace and the system will take over the throttle of the vehicle to maintain the same speed. Fig.3 shows its state machine designed by SCADE Suit. Esterel Technologies provides this case in its SCADE suit and also develops many requirements about this case. Here we list three safety-critical requirements that we have to verify:

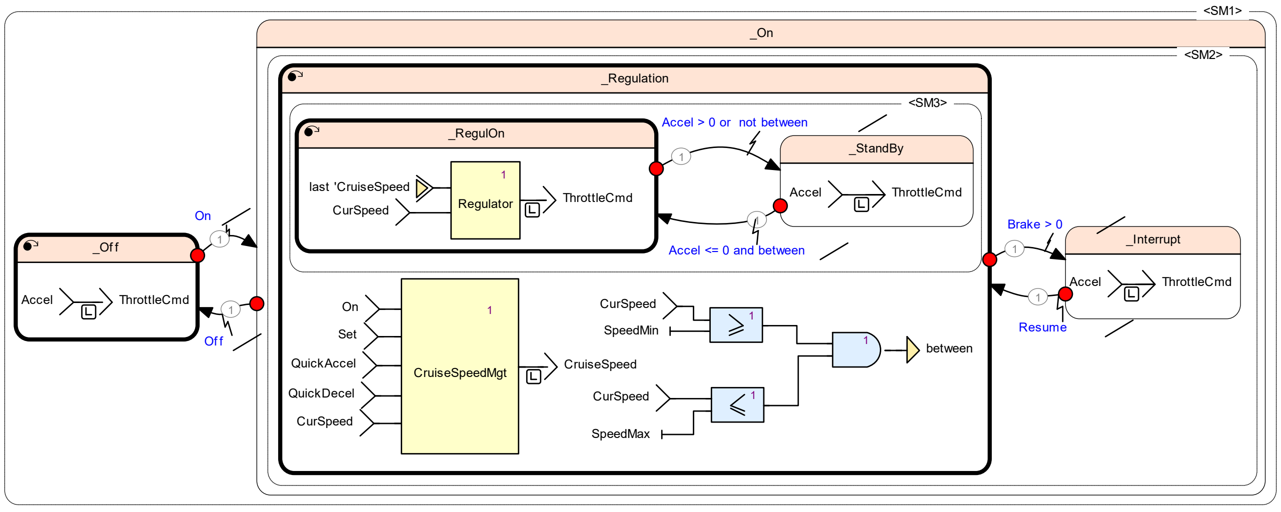


Figure 3: Cruise Control System Designed by SCADE Suit.

- Requirement 1: If Cruise control is always off, the throttle command is always controlled by the accelerator pedal sensor.
- Requirement 2: When the car is under the cruise control, if driver brakes, the system will always be interrupted or off.
- Requirement 3: All paths in the system satisfy that if the driver disables the cruise control, the next state of the system will be Off for all paths.

Verifying these requirements with Scade2Nu. SCADE suit supports users to convert a graphical model into a textual SCADE language program. We also need to create a new file named `VarBound` which sets the bound of variables for the simple abstraction. Scade2Nu project accepts these two files and achieves the target `.smv` model by clicking the translation button in the toolbar. In this case, we get the `CruiseControl.smv` model. We present the NuSMV model file online[4]. Next, we write down the CTL or LTL formulas that

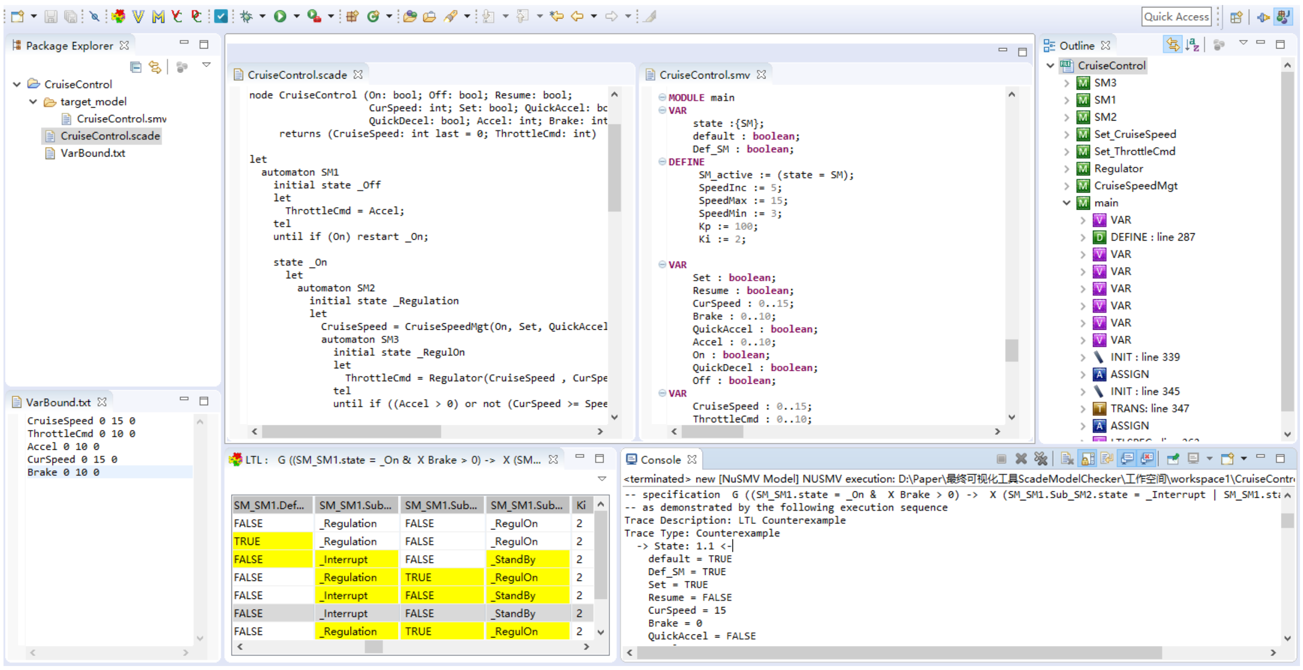
[4]Scade2Nu homepage and more cases, see https://sites.google.com/view/scade2nu

Table 1: Requirements Specifications in NuSMV.

RQ	Specifications in Temporal Logics	Feedback
1	`G((SM_SM1.state = _Off & G !On) -> Accel = next(ThrottleCmd))`	True
2	`G(SM_SM1.state = _On & X Brake > 0 -> X (SM_SM1.Sub_SM2.state = _Interrupt | SM_SM1.state = _Off))`	False
3	`AG(Off -> AX SM_SM1.state = _Off)`	False

correspond to system requirements. Then we can verify them through NuSMV. Fig.4 shows the graphic view of Scade2Nu and Table 1 presents the formulas written with temporal specifications in NuSMV.

Evaluation and limitations. Scade2Nu provides a counterexample visualizer (the middle bottom of Fig.4) that shows traces in a more friendly manner than the simple text produced by NuSMV because of the combination with NuSeen[AGR17]. Therefore, it can debug the SCADE model by analyzing the detail results. In this case, NuSMV reports that requirement 1 is valid. Requirement 2 and 3 are false and demonstrated by the generated counterexamples. For instance, we find the unusual transition that from state `_Interrupt` to `_Regulation` in `SM2` when the driver brakes. In another word, the circumstance that cruise control system will not always interrupt when the driver brakes may happen. So we have to modify it in SCADE model: the guard of transition from state `_Interrupt` to `_Regulation` need to be `Resume and not (Brake > 0)`. The verification time of Scade2Nu depends on the right bounds of variables. We firstly did not set variable bounds and it took many hours to get the results. After we adding the bound option, the time reduces to few seconds and we get the same results. Therefore one of the limitations of Scade2Nu now is that users need to set variable bounds manually.



```
node CruiseControl (On: bool; Off: bool; Resume: bool;
                    CurSpeed: int; Set: bool; QuickAccel: bc
                    QuickDecel: bool; Accel: int; Brake: int
     returns (CruiseSpeed: int last = 0; ThrottleCmd: int)

let
  automaton SM1
    initial state _Off
    let
      ThrottleCmd = Accel;
    tel
    until if (On) restart _On;

    state _On
      let
        automaton SM2
          initial state _Regulation
          let
            CruiseSpeed = CruiseSpeedMgt(On, Set, QuickAccel
            automaton SM3
              initial state _RegulOn
              let
                ThrottleCmd = Regulator(CruiseSpeed , CurSpe
              tel
              until if ((Accel > 0) or not (CurSpeed >= Spee
```

```
MODULE main
VAR
    state :{SM};
    default : boolean;
    Def_SM : boolean;
DEFINE
     SM_active := (state = SM);
     SpeedInc := 5;
     SpeedMax := 15;
     SpeedMin := 3;
     Kp := 100;
     Ki := 2;

VAR
    Set : boolean;
    Resume : boolean;
    CurSpeed : 0..15;
    Brake : 0..10;
    QuickAccel : boolean;
    Accel : 0..10;
    On : boolean;
    QuickDecel : boolean;
    Off : boolean;
VAR
    CruiseSpeed : 0..15;
    ThrottleCmd : 0..10;
```

```
CruiseControl
  SM3
  SM1
  SM2
  Set_CruiseSpeed
  Set_ThrottleCmd
  Regulator
  CruiseSpeedMgt
  main
    VAR
    DEFINE : line 287
    VAR
    VAR
    VAR
    VAR
    VAR
    VAR
    INIT : line 339
    ASSIGN
    INIT : line 345
    TRANS: line 347
    ASSIGN
```

```
CruiseSpeed 0 15 0
ThrottleCmd 0 10 0
Accel 0 10 0
CurSpeed 0 15 0
Brake 0 10 0
```

LTL : G ((SM_SM1.state = _On & X Brake > 0) -> X (SM...

SM_SM1.Def...	SM_SM1.Sub...	SM_SM1.Sub...	SM_SM1.Sub...	Ki
FALSE	_Regulation	FALSE	_RegulOn	2
TRUE	_Regulation	FALSE	_RegulOn	2
FALSE	_Interrupt	FALSE	_StandBy	2
FALSE	_Regulation	TRUE	_RegulOn	2
FALSE	_Interrupt	FALSE	_StandBy	2
FALSE	_Interrupt	FALSE	_StandBy	2
FALSE	_Regulation	TRUE	_RegulOn	2

```
<terminated> new [NuSMV Model] NUSMV execution: D:\Paper\最终可视化工具ScadeModelChecker\工作空间\workspace1\CruiseContr
-- specification  G ((SM_SM1.state = _On &  X Brake > 0) ->  X (SM_SM1.Sub_SM2.state = _Interrupt | SM_SM1.sta
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    default = TRUE
    Def_SM = TRUE
    Set = TRUE
    Resume = FALSE
    CurSpeed = 15
    Brake = 0
    QuickAccel = FALSE
```

Figure 4: The graphic view of Scade2Nu.

4 Conclusion and Future Work

In this paper, we have presented Scade2Nu, and it provides a new framework for the requirements verification of SCADE models. For simple safety requirements, Scade2Nu users only need to write down the temporal specifications instead of building a monitor SCADE model in Design Verifer. For complex temporal properties that DV cannot express, Scade2Nu has the capability of expression and verification. Additionally, the results of the verification are more detailed because counterexamples are one of the strengths of model checkers.

However, Scade2Nu now only concentrates on the state-based control systems, so we plan to extend it to support systems that are data-flow oriented. Scade2Nu also needs to be tested by more industrial cases. Therefore, for the purpose of verifying more complex SCADE models, we also plan to support more temporal operations, arrays, as well as high-order operations of SCADE language in future work.

4.0.1 Acknowledgements

This work is partially supported by National Natural Science Foundation of China (No. 61602178), Shanghai Science and Technology Committee Rising-Star Program (No.18QB1402000), China HGJ Project under Grant (No. 2017ZX01038 102-002) and National Defense Basic Scientific Research Program of China (No. JCKY2016204B503).

References

[ADS$^+$04] Parosh Aziz Abdulla, Johann Deneux, Gunnar Stålmarck, Herman Ågren, and Ove Åkerlund. Designing safe, reliable systems using scade. In *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, pages 115–129. Springer, 2004.

[AGR17] Paolo Arcaini, Angelo Gargantini, and Elvinia Riccobene. Nuseen: a tool framework for the nusmv model checker. In *Software Testing, Verification and Validation (ICST), 2017 IEEE International Conference on*, pages 476–483. IEEE, 2017.

[AY01] Rajeev Alur and Mihalis Yannakakis. Model checking of hierarchical state machines. *ACM Transactions on Programming Languages and Systems*, 23(3):273–303, 2001.

[CH00] Edmund M Clarke and Wolfgang Heinle. Modular translation of statecharts to smv. Technical report, Citeseer, 2000.

[DHM11] Ilyas Daskaya, Michaela Huhn, and Stefan Milius. Formal safety analysis in industrial practice. In *International Workshop on Formal Methods for Industrial Critical Systems*, pages 68–84. Springer, 2011.

[Dor08] Francois-Xavier Dormoy. Scade 6: a model based solution for safety critical software development. In *Proceedings of the 4th European Congress on Embedded Real Time Software (ERTS'08)*, pages 1–9, 2008.

[ET15] Inc Esterel Technologies. Scade language - reference manual 2.1. 2015.

[Har87] David Harel. Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274, 1987.