Constraint-based Software Specifications for Safety-critical Applications

Chia-Kuo Wu and Chin-Feng Fan
Dept. of Computer Science and Engineering Yuan-Ze University, Chung-Li, Taiwan
csfanc@saturn.yzu.edu.tw

Abstract-*Constraint-based specifications add constraints in behavior or properties specifications. If these constraints are violated at run time, appropriate adjustment can be made. This research extends SpecTRM-RL, a newest constraint-based specification language developed by Nancy Leveson, to incorporate graphic representation, augmented assertions, and safety analysis. We also convert portions of the specifications to DFM (Dynamic Flowgraph Methodology) to generate timed fault trees systematically at design stage. These techniques enhance the quality and safety of safety-critical software.*
Keywords: constraints, SpecTRM-RL, DFM, fault tree, safety analysis.

1. Introduction

Constraint-based specifications refer to the specifications that add constraints in behavior modeling or properties specification. They are useful for safety-critical computing systems, such as computing systems in the areas of aviation, transportation, medicine, and nuclear power plants. If the pre-specified constraints are violated at run time, appropriate adjustment can be made to ensure system safety. In short, constraint-based specification can help operators to monitor whether the system is in unsafe situation. This research aims to develop a constraint-based software specification approach suitable for safety-related applications. Such specifications should support both run-time hazard detection and static hazard/safety analysis.

SpecTRM-RL[6] is a newest constraint-based specification language developed by Nancy Leveson for safety-related applications. This research proposes to extend SpecTRM-RL to improve its visual representation, completeness and consistency of constraints, as well as safety analysis so as to enhance system safety.

2. Background

2.1. SpecTRM-RL

SpecTRM[6] (Specification Tools and Requirements Methodology) is a software specification method based on Intent structure [5]. Both are developed by Nancy Leveson. SpecTRM-RL is SpecTRM's constraint-based Requirements Language. The language is used to express the blackbox behavior at Level 3 of the Intent system structure [5]. SpecTRM-RL uses a state machine model. Its system diagram includes a supervisory mode, inferred system states, input/output messages and hardware devices. AND/OR tables are used to represent the logic of conditions in different outputs, inputs and state transitions. Note that the boolean values in a column of the AND/OR tables are "AND" together and those in different columns are "OR" together. Most of the default constraints are related to values, ranges, types, timing, and capacity; these constraints are expressed in natural language. SpecTRM-RL's system diagram, constraints, and AND/OR tables are shown in Figures 1 to 3.

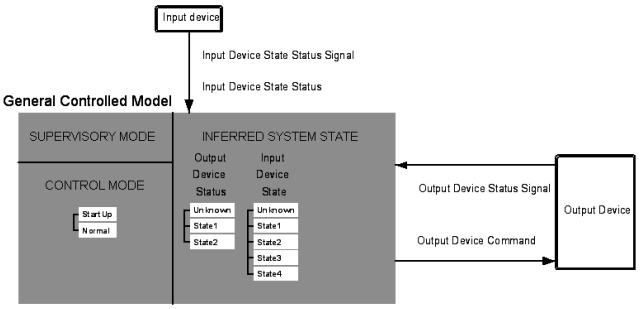


Figure 1. System diagram in SpecTRM-RL

Destination: Output Device
Message: Output Device Command Data
Timing Behavior:
Initiation Delay: 10 milliseconds
Completion Deadline: 20 milliseconds
Exception-Handling: Send MSG to the management
Feedback Information: Send MSG to Controller
Variables: Output Device Status Signal
Values: {Command}
Relationship: Should be {State2} after Controller sends signal to Output Device to be State2
Min. time (latency): 1 seconds
Max time: 6 seconds
Reversed By: The Output Device is State1 by the Other command
Description: The Output Device Command causes the Output Device to be State2
Comments: If the Output Device is State2 , issuing the second lower command will have no effect
References: Output Device Status , Input Device State

Figure 2. SpecTRM-RL constratins

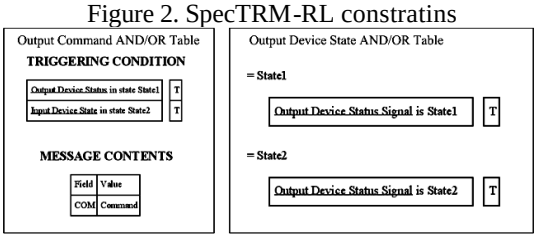
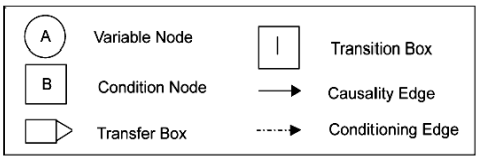


Figure 3. AND/OR tables

2.2. DFM

Dynamic Flowgraph Methodology (DFM) [1,5] developed by George E. Apostolakis is an approach to modeling and analysis of the safe behavior of both hardware and software in a computer-controlled system. Software can be expressed in detailed requirements or design specifications. The methodology provides a systematical way to generate timed fault trees from DFM specification for safety analysis. Our research proposes to incorporate DFM to expand a portion of the specifications into design details so as to develop fault trees for analysis. DFM's system graph is shown in Figure 4.



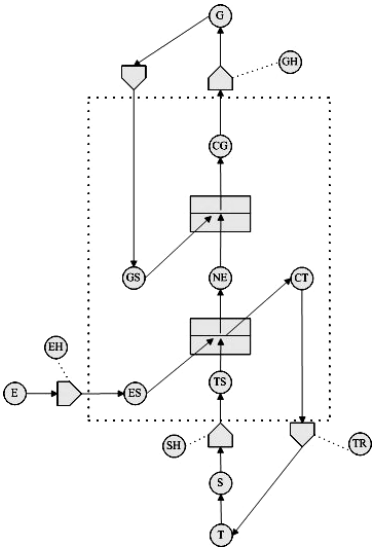


Figure 4. DFM System graph

2.3. Fault Tree Analysis

Fault tree analysis (FTA) is a popular technique used to statically analyze system safety in a backward fashion. It involves specifying an undesired event as the top event to analyze, followed by identifying all of the associated elements in the system that could cause that top event to occur. It is extensively used in aviation, nuclear energy, and the electronics industry. Fault tree analyses are performed graphically using a logical structure of AND and OR gates. Figure 5 is a simple fault tree model. Timed fault trees refers to the trees generated backward in a level by level fashion, and each level happens at a different time, say t, t-1, etc.

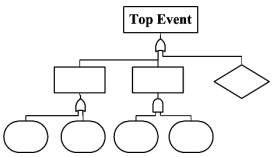


Figure 5. A fault tree

3. Our approach

SpecTRM-RL, with its associated tool and Intent framework, is a promising method appropriate for safety-critical applications. However, since SpecTRM-RL is still under development, many aspects can be enhanced. For instance, the system diagram is the only required diagram. More graphic representation will be desired to improve the readability of the specifications. The completeness, consistency, and priorities of constraints should be further examined. Besides run-time monitoring using constraints as safety guards, we suggest to add template-based fault tree analysis to provide an extra level of safety assurance. We extend SpecTRM-RL in the steps shown in Figure 6.

Steps 1.1 and 1.2 in Figure 6 add graphic representation to the method. Statecharts [2] can be added to show state transitions; UML sequence diagrams can be used to explicitly show the interaction between input devices, output devices and software commands. On the other hand, it is also possible to automatically generate statecharts from the given AND/OR tables if the state that a transition originates from is indicated in the AND/OR table. Figure 7 gives a sample case of this automatic conversion from AND/OR tables to a statechart.

The completeness of constraints is considered in Step 2. To make the constraint set more complete, we add several categories of assertions focusing on relations among variables and states. In general, both constraints and assertions refer to statements of relations that we think to hold; yet, assertions are usually used in program contexts. Here we use them interchangeably. We propose a formal expression for constraints so that they can be checked automatically. Step 3 checks the consistency among constraints/assertions. However, there may be too many constraints to check efficiently. Thus, Step 4 provides a simple way to identify important AND/OR tables, and consequently their constraints should have higher priorities.

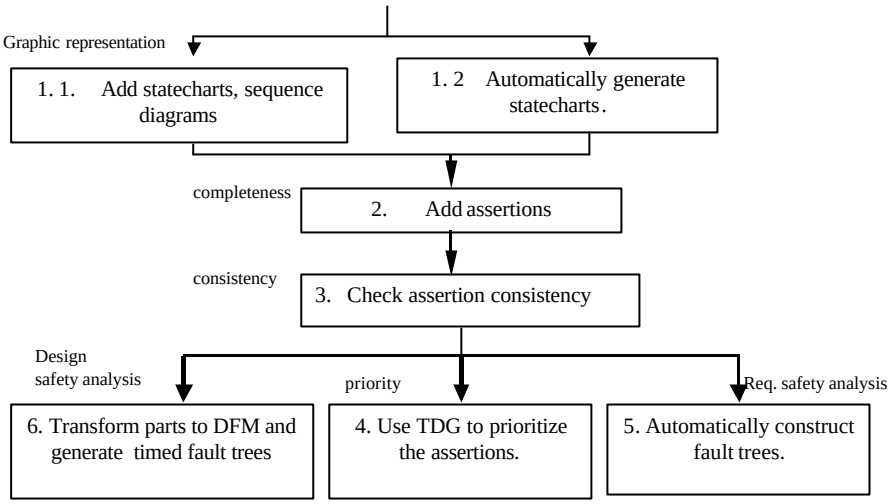
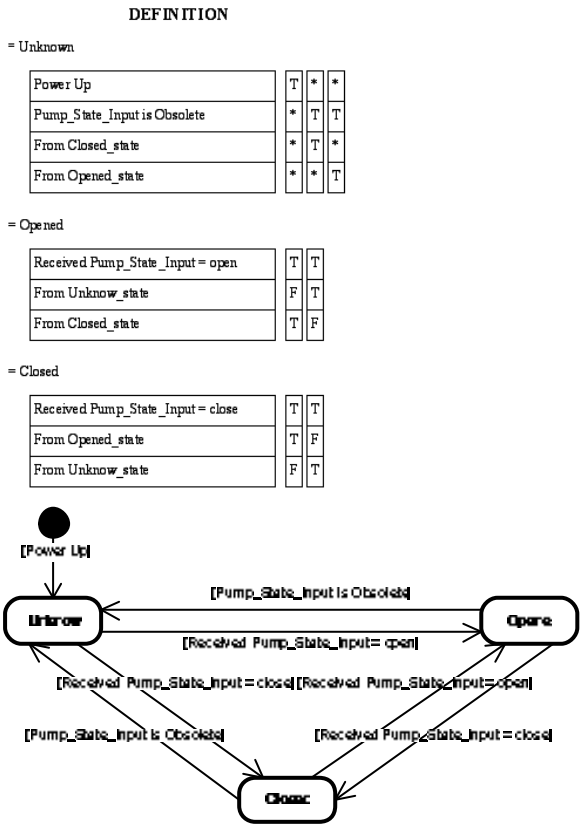


Figure 6. Our Steps



emphasize relations between different variables. We believe that the relations between variables, devices, and different kinds of timing are particularly important. Thus, we add assertions to SpecTRM-RL so that the specified relations can be checked at run-time. If the relations do not hold, the operator will be notified. The suggested assertions are as follows:

1. Reverse checking of state transitions:
   For critical states, the correctness of the state transition can be checked by reverse checking. The following is an example where (-) refers to the previous state:
   Level_State =Too_High → (-)(Level_State = Abnormal_High) ^ (Water_level >M2)

2. Checking of I/O boundaries, ranges, timing, and formats:
   The following is an example:
   (Time since Pump_State = opened was last Received > 5 sec)^ (Water_level ≧ (-)(Water_level)) ^ (0 < Water_level < MaxWater C)

3. Invariants：
   Invariants specify the relations that always hold among process variables and devices, including physical invariants [8]. An example is given below:
   (Steam s) ≦ (MaxSteam W)

   Formulae to calculate process variables can also be viewed as invariants. For example, water quantity change is
   $dq/dt = p_e * P - v_e * V - s_e$
   where p_e, v_e, s_e are status of the pump, valve, and steam, respectively; while P, V are the amount of water flow of the pump and the valve.

4. Precondition: The precondition of an action should be checked.

5. Failure conditions:
   Conditions explaining why the system faultily enters the specified states can also be specified in AND/OR tables. They are used to generate fault trees. The tables indicating "fault of Gate_Raised" and "fault of ate_Lower" in Figure 8 are such examples.

6. Recovery assertions: The original SpecTRM-RL has constraints for reverse action. We elaborate and specify them in a formal form, which may relate several variables/tables, for recovery action once the related failures occur.

The consistency checking of constraints is not mentioned in the original SpecTRM-RL. We provide an algorithm to check consistency in timing between related outputs and inputs, as well as between feedback and output commands. The Algorithm is shown in Figure 9.

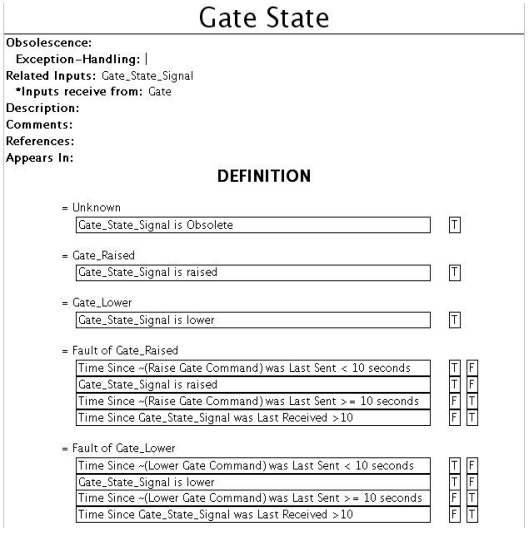


Figure 8. Fault conditions added

```
BEGIN
 FOR all Outputs
  Check the constraint - Feedback Information in Output
   IF there is empty, it means that there has no feedback information for this Output.
     Check next Output.
   IF it is not empty
     In the same model, find Input title which is the same as Feedback Information.
       Check the constraint - Source in Input. If it is not the same as the Destination
         of Output, constraints are not consistent.
       Check constraint - Possible Values in Input. If it is not the same as the Values
         of Output, constraints are not consistent.
       Check the constraint - Related Outputs in Input. If it does not include Output
         title, constraints are not consistent.
       Check time definition in Output and Input, and it must satisfy the formula :
         Min. time (latency) = Minimum Time Between Outputs + Minimum Time
         Between Inputs
 END FOR
END
```

Figure 9. Constraint consistency checking

5. Prioritize Assertions

With the original constraints and the augmented assertions mentioned above, there are too many constraints/assertions to check efficiently at run time. We have developed a method to estimate the importance of an AND/OR table. Then, the constraints/assertions related to important AND/OR tables should be checked first.

We first define Table Dependence Graph (TDG). We may draw the dependency diagram for variables in the predicate specified by an AND/OR table of an output command. Such a graph is called TDG. To prioritize constraints, we first identify critical failures as the top events and draw the related fault trees. Then, we locate the "software code error" nodes in the leaves of these fault trees. The results of these software errors are incorrect output commands shown at one level above in the fault tree. We then construct the TDG for the logic specified by the AND/OR table of the incorrect output commands. This is shown in Figure 10 where triangles represent fault trees. The weight of each fault tree will be added to its connected nodes in the graph and then summed up to get the priorities of the involved AND/OR Tables (i.e., variables and devices). Assertions of the

more important tables have higher priorities than those of the less important ones.

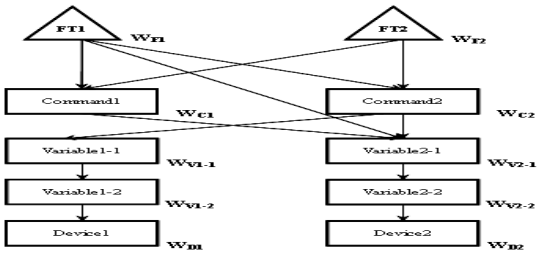


Figure 10. A TDG sample

6. Safety Analysis Using Fault Trees

We propose to associate static safety analysis with the extended specification. Since the extended specifications include "failure conditions" information in its AND/OR tables, we have developed a systematic way to convert the extended SpecTRM-RL specification into a fault tree based on a given template for an undesired event. We consider many possible faults such as hardware errors, message delay, computing errors, encoding errors and so on. This requirement- stage fault tree template is shown in Fig. 11.

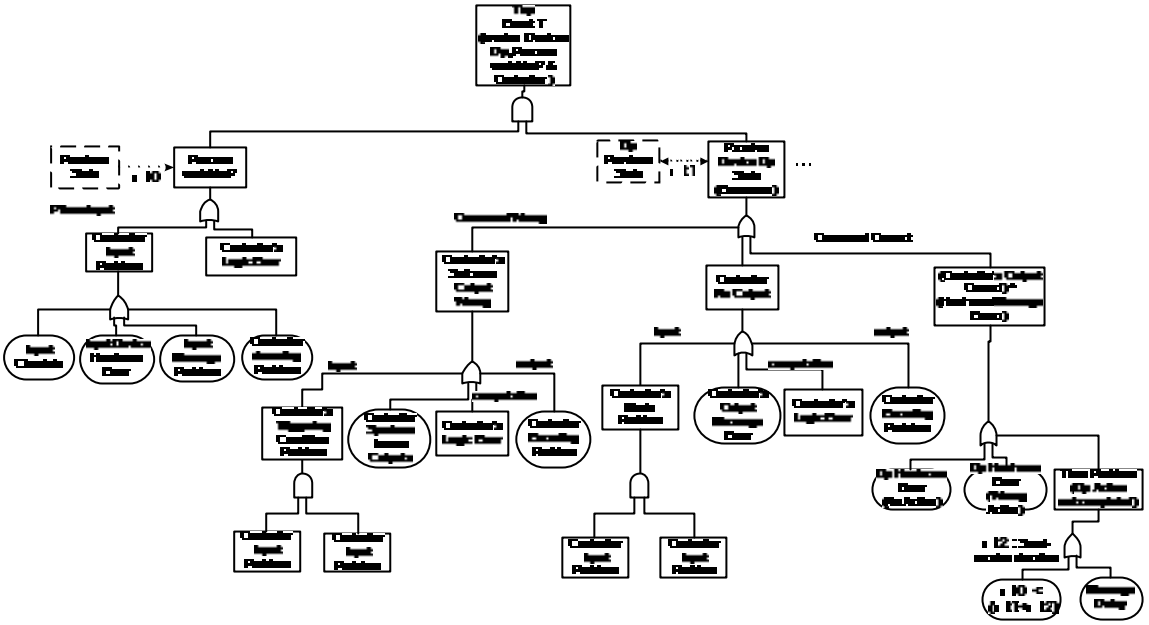
Besides, we have combined DFM to generate design-level fault trees. We have studied and compared several popular and promising specification methods using tables and related to safety-critical domains. These methods include SpecTRM-RL, DFM [4] and SCR [3]. We concluded that an ideal approach to design-level safety verification should take the advantages of both SpecTRM-RL and DFM because DFM provides a systematic fault tree construction method. We first developed a method that transform SpecTRM-RL specification into a DFM's graph. Designers can then fill in the details of DFM's decision tables, i.e., software functions. However, if the detailed control logic has already defined in SpecTRM-RL, we can then translate the AND/OR tables into DFM decision tables. We first identify an important output command. Then we check triggering conditions of the command from its AND/OR table. The involved State values (or internal values) and Input messages in the triggering condition will be taken and drawn as DFM's nodes to generate DFM graph from SpecTRM-RL in a backward fashion. As an example, the specification of the pump command in a Steam-boiler case is transformed into DFM and shown in Figure 12. Note that the decision table in the figure expresses the software logic extracted from the AND/OR table of SS (steam state), given the input SV (steam quality). The details of this case can be found in [7]. The timed fault tree of DFM can be then generated for safety analysis.

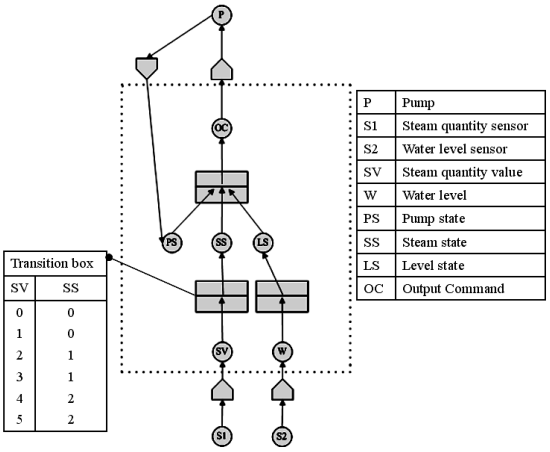
The major difference between the proposed fault trees in Fig. 11 and the ones generated from the converted DFM is not only stage difference (requirements and design stages). DFM's fault tree generation focuses on software part and can be performed systematically on a step-by-step base. But it only considers the missing of the correct behavior of modeled components. On the other hand, our template-based fault trees not only consider the causes due to modeled components, but also extend to include other possible faults, such as message problems or software unintended functions.

Results of the suggested safety analysis using fault trees can work as feedback to re-design and to check the completeness of constraints. If the relations leading to an undesired event are missing in the original constraint pool, they can be added. Results of safety analysis can also be used to judge constraint priorities, or used at run time to identify the violated relations once potential hazards occur. We have successfully applied the above extensions to two cases, a simple steam-boiler case and a railroad-crossing case [7].

7. Conclusion

We extended SpecTRM-RL, Nancy Leveson's newest requirements specification method. This research added graphic representation to SpecTRM-RL to improve its understandability. We also augmented constraints by several categories of assertions to make the constraint pool more complete and effective. Moreover, static analysis using fault trees are generated systematically from the specification to add an extra level of safety guards besides the dynamic constraint checking at run time. The preliminary results are encouraging. An associated tool is under construction and more realistic cases are currently being examined. The results will be reported shortly.





Transition box	
SV	SS
0	0
1	0
2	1
3	1
4	2
5	2

P	Pump
S1	Steam quantity sensor
S2	Water level sensor
SV	Steam quantity value
W	Water level
PS	Pump state
SS	Steam state
LS	Level state
OC	Output Command