A Formal Semantics for Dynamic Fault Trees

September 7, 2002
Version 0.1

David Coppit
Dept. of Comp. Sci.
The College of William and Mary
Williamsburg, VA 23185
david@coppit.org

Kevin J. Sullivan
Dept. of Comp. Sci.
University of Virginia
Charlottesville, VA 22903
sullivan@cs.virginia.edu

Joanne Bechta Dugan
Dept. of Elec. Eng.
University of Virginia
Charlottesville, VA 22903
jbd@ee.virginia.edu

Contents

1 Scope

This document formally specifies the abstract syntax and semantics of dynamic fault trees. The specification is written in Z [1], with the semantics expressed in terms of a lower-level domain called *fault tree automata*. A subset of this lower-level domain is then formally defined in terms of the well-understood domain of Markov chains. Complete specification of the semantics of fault tree automata will require the use of additional low-level domains besides Markov chains, and is not covered in this document.

While this document formalizes the key aspects of the dynamic fault tree framework, there are several related aspects that are not covered. Most obvious is the formal semantics of subsets of DFTs which can not be expressed as Markov chains. Another is a divide-and-conquer technique for modularizing a DFT, solving the modules independently, and integrating the results. Similarly, we do not address the formal semantics of DFTs with regard to properties of interest besides unreliability, such as the sensitivity of components or the modeling of systems that operate in multiple phases.

2 Conventions and Notation

This section defines the conventions and text styles used throughout this document. The notation and convention descriptions specific to the Z notation have been omitted.

defined term: A defined term is underlined.

variableName: Variable names begin with a lower case letter. Additional words in the variable name begin with capital letters and are concatenated.

TypeName: Type names begin with an upper case letter. Additional words in the type name begin with capital letters and are concatenated.

3 Definitions

basic event: A basic event models either the failure of an unelaborated subsystem, or the occurrence of some phenomenon that affects the system.

gate: A gate models some combination or sequence of event occurrences.

constraint: An invariant imposes a constraint on the occurrence of events in the model.

event: An event models the occurrence of some phenomenon that affects the system, or the failure of a system, subsystem, or component of a system. Events can be either basic events or gates.

causal basic event: A basic event, the occurrence of which can cause the occurrence of all other newly occurred events in a fault tree.

AND gate: A gate whose output event is occurred if all of the input events are occurred.

OR gate: A gate whose output event is occurred if any of the input events are occurred.

threshold gate: A gate whose output event is occurred if the number of input events that are occurred exceeds a specified threshold.

priority-and (PAND) gate: A gate whose output event is occurred if all of the input events have occurred and if they occurred "in order".

spare gate: A gate in which spare inputs are used in order until no operational input is available, in which case the event associated with the output of the gate occurs.

functional dependency (FDEP) constraint: A constraint which specifies that the dependent events must occur if the trigger event occurs.

sequence (SEQ) constraint: A constraint which specifies that the input events can only occur "in order".

coverage model: Three values used to model the probability that either a basic event occurs but is not visible to the system, a basic event occurs and can be handled by the system, or a basic event occurs and results in a failure.

dormancy: A factor between 0 and 1 inclusive that is used to attenuate a spare when it is not in use.

in order occurrence: Two events A and B are said to occur in order if A occurs before *or at the same time* as B.

fault tree state: the state of a fault tree, consisting of the number of occurrences for each event, the allocation of spares, the history of event occurrences, and the fault tree status.

history: a sequence of event states resulting from a sequence of event occurrences.

time step: one position in a fault tree history

4 Basic Types

We begin by defining an abstract system of real numbers and operations in this section.

In this section we begin the formal specification with the definition of the abstract syntax of DFTs in Z. We first define an abstract system of real numbers and operations.

$$[\mathbb{R}]$$

$$0_{\mathbb{R}} : \mathbb{R}$$
$$1_{\mathbb{R}} : \mathbb{R}$$

We introduce $\mathbb{R}$ as a given type, and declare $0_{\mathbb{R}}$ and $1_{\mathbb{R}}$ to be elements of that type. In our use of real numbers, the subscript is used to distinguish between the values and operators used for non-reals and those used for reals.

$$\_ +_{\mathbb{R}} \_ : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$$
$$\_ *_{\mathbb{R}} \_ : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$$
$$\_ /_{\mathbb{R}} \_ : \mathbb{R} \times \mathbb{R} \nrightarrow \mathbb{R}$$
$$\_ =_{\mathbb{R}} \_ : \mathbb{R} \leftrightarrow \mathbb{R}$$
$$\_ \neq_{\mathbb{R}} \_ : \mathbb{R} \leftrightarrow \mathbb{R}$$
$$\_ <_{\mathbb{R}} \_ : \mathbb{R} \leftrightarrow \mathbb{R}$$
$$\_ >_{\mathbb{R}} \_ : \mathbb{R} \leftrightarrow \mathbb{R}$$
$$\_ \leq_{\mathbb{R}} \_ : \mathbb{R} \leftrightarrow \mathbb{R}$$
$$\_ \geq_{\mathbb{R}} \_ : \mathbb{R} \leftrightarrow \mathbb{R}$$

$$+/_{\mathbb{R}} : \mathbb{F}\,\mathbb{R} \rightarrow \mathbb{R}$$

$$\_ *_{f(\mathbb{R})} \_ : (\mathbb{R} \rightarrow \mathbb{R}) \times \mathbb{R} \rightarrow (\mathbb{R} \rightarrow \mathbb{R})$$
$$\_ +_{f(\mathbb{R})} \_ : (\mathbb{R} \rightarrow \mathbb{R}) \times (\mathbb{R} \rightarrow \mathbb{R}) \rightarrow (\mathbb{R} \rightarrow \mathbb{R})$$
$$\_ *_{pf(\mathbb{R})} \_ : (\mathbb{R} \nrightarrow \mathbb{R}) \times \mathbb{R} \rightarrow (\mathbb{R} \nrightarrow \mathbb{R})$$
$$\_ +_{pf(\mathbb{R})} \_ : (\mathbb{R} \nrightarrow \mathbb{R}) \times (\mathbb{R} \nrightarrow \mathbb{R}) \rightarrow (\mathbb{R} \nrightarrow \mathbb{R})$$

$$intToReal : \mathbb{Z} \rightarrow \mathbb{R}$$

$$intToReal\, 0 = 0_{\mathbb{R}}$$
$$intToReal\, 1 = 1_{\mathbb{R}}$$
$$\mathrm{dom}(\_ /_{\mathbb{R}} \_) = \mathbb{R} \times \mathbb{R} \setminus \{0_{\mathbb{R}}\}$$
$$\forall x, y : \mathbb{Z} \bullet intToReal\, x = intToReal\, y \Leftrightarrow x = y$$

We introduce type definitions for functions that operate on real numbers and functions of real numbers, abstracting the definitions. We first declare addition and division as an infix functions from pairs of reals to reals. Next we define the "distributed summation" operator, which computes the sum of a finite set of reals. We then define four operators for performing the distributed summation and product of both total and partial functions on reals. The function *intToReal* is used to map integers to reals, similar to a cast in programming languages.

$$Boolean ::= True \mid False$$

We define a Boolean type.

$$Probability == \{p : \mathbb{R} \mid 0_{\mathbb{R}} \leq_{\mathbb{R}} p \leq_{\mathbb{R}} 1_{\mathbb{R}} \bullet p\}$$

We define a probability as a real number between the values of 0 and 1 inclusive.

$$Rate == \{ r : \mathbb{R} \mid r \geq_{\mathbb{R}} 0_{\mathbb{R}} \bullet r \}$$

We define a rate as a real number greater than or equal to 0.

$$Time == \{ t : \mathbb{R} \mid t >_{\mathbb{R}} 0_{\mathbb{R}} \bullet t \}$$

We define time to be a real number greater than 0.

5 Abstract Syntax of Fault Trees

5.1 Event Identifiers and Events

$[Event]$

Event is a given type that represents failures or event occurrences in the fault tree.

5.2 Basic Events

BasicEvents

$$basicEvents : \mathbb{F}\, Event$$

The basic events of a system are represented as a finite set of events.

5.3 Gates

In this section we specify the abstract syntax of the gates of a dynamic fault tree.

Gates

$$andGates : \mathbb{F}\, Event$$
$$orGates : \mathbb{F}\, Event$$
$$thresholdGates : \mathbb{F}\, Event$$
$$pandGates : \mathbb{F}\, Event$$
$$spareGates : \mathbb{F}\, Event$$
$$gates : \mathbb{F}\, Event$$
$$thresholds : Event \pfun \mathbb{N}_1$$

$$\langle andGates, orGates, thresholdGates, pandGates, spareGates \rangle \quad (5.1)$$
$$\quad \text{partition } gates$$
$$\mathrm{dom}\, thresholds = thresholdGates \quad (5.2)$$

This schema defines the gates of a fault tree as finite sets of events, one set for each type of gate. Line 5.1 states that these sets partition the set of all gates in the fault tree. As stated on line 5.2, each threshold gate has an associated non-zero natural number that represents the threshold value. A threshold gate occurs if the number of occurred input replicates is greater than or equal to the *threshold* value.

5.4 Constraints

$$InputSequence == \text{iseq}\, Event$$

We define an input sequence as simply a sequence of events which does not contain repeated elements. This definition will be used in the definition of the inputs of gates and in the definition of the constraints.

Constraints

$$seqs : \mathbb{F}\, InputSequence$$
$$fdeps : Event \rightarrow InputSequence$$

The schema above defines a sequence enforcer as an constraint over non-empty sequences of events. A functional dependency is a partial function from events to non-empty sequences of events. In the predicate we overspecify by defining the dependent events as a sequence instead of a set. We do this to improve readability later in the specification.

5.5 Fault Trees

Having specified the basic events, gates, and constraints, in this section we present the full specification of the fault tree abstract syntax.

$$InputsMap == Event \pfun InputSequence$$

First we define a type for mapping a gate to its inputs. This function is partial because basic events are events, but do not have inputs.

$$\begin{array}{|l} IsDirectlyInputTo\_ : \mathbb{P}(Event \times Event \times InputsMap) \\ \hline \forall from, to : Event;\ inputs : InputsMap \mid to \in \operatorname{dom} inputs \bullet \\ \quad IsDirectlyInputTo(from, to, inputs) \Leftrightarrow from \in \operatorname{ran}(inputs\, to) \end{array}$$

IsInputDirectlyTo is true if the "from" event is in the inputs list of the "to" event. The three arguments for this function are the "from" event, the "to" event, and the partial function mapping gates to their inputs.

$$\begin{array}{|l} IsInputTo\_ : \mathbb{P}(Event \times Event \times InputsMap) \\ \hline \forall from, to : Event;\ inputs : InputsMap \mid to \in \operatorname{dom} inputs \bullet \\ \quad IsInputTo(from, to, inputs) \Leftrightarrow \\ \qquad IsDirectlyInputTo(from, to, inputs) \lor \\ \qquad (\exists g : \operatorname{dom} inputs \bullet from \in \operatorname{ran}(inputs\, g) \land IsInputTo(g, to, inputs)) \end{array}$$

IsInputTo is true if either the *from* event is directly input to the *to* event, or if there is some gate to which *from* is an input and which is an input (recursively) to *to*. The three arguments for this function are the "from" event, the "to" event, and the partial function mapping gates to their inputs.

$$ReplicationMap == Event \fun \mathbb{N}_1$$

We also define a replication function which maps events to their replications.

$$\begin{array}{|l} NumberOfReplicatesInInputs : InputSequence \times ReplicationMap \fun \mathbb{N} \\ \hline \forall is : InputSequence;\ rs : ReplicationMap \bullet \\ \quad NumberOfReplicatesInInputs(is, rs) = \\ \qquad \textbf{if } is = \langle\rangle \textbf{ then } 0 \\ \qquad \textbf{else } rs(head\, is) + NumberOfReplicatesInInputs((tail\, is), rs) \end{array}$$

This helper function, given a sequence of events and a replication mapping, determines the total number of event replicates in the sequence.

FaultTree

$$
\begin{array}{l}
BasicEvents \\
Gates \\
Constraints \\
\\
events : \mathbb{F}\, Event \\
inputs : InputsMap \\
replications : ReplicationMap \\
\hline
\langle basicEvents, gates \rangle \text{ partition } events \qquad (5.3) \\
\\
\mathrm{dom}\, inputs = gates \qquad (5.4) \\
\forall g : gates \bullet \mathrm{ran}(inputs\, g) \subseteq events \\
\forall g : gates \bullet \neg\, IsInputTo(g, g, inputs) \\
\\
\forall sg : spareGates \bullet \mathrm{ran}(inputs\, sg) \subseteq basicEvents \qquad (5.5) \\
\forall sg : spareGates;\ be : basicEvents \mid IsDirectlyInputTo(be, sg, inputs) \bullet \\
\quad \neg\, (\exists g : gates \setminus spareGates \bullet IsDirectlyInputTo(be, g, inputs)) \\
\\
\forall s : seqs \bullet \mathrm{ran}\, s \subseteq events \qquad (5.6) \\
\\
\mathrm{dom}\, fdeps \subseteq events \qquad (5.7) \\
\mathrm{dom}\, replications = events \\
\forall t : \mathrm{dom}\, fdeps \bullet replications\, t = 1 \\
\forall is : \mathrm{ran}\, fdeps \bullet \mathrm{ran}\, is \subseteq basicEvents \\
\\
\forall g : gates \bullet replications\, g = 1 \qquad (5.8)
\end{array}
$$

The *FaultTree* schema defines the syntactic structure of a fault tree. The *events* set is the set of all events in the fault tree. *inputs* is a mapping for the inputs of each gate in the fault tree, and *replications* is a similar mapping for the replications of the basic events.

The constraints state the following:

- (5.3) An event is either a basic event, an AND gate, an OR gate, a threshold gate, a PAND gate, or a spare gate.
- (5.4) Only gates can have inputs, the inputs must be one of the events in the fault tree, and no gate can be input to itself (directly or indirectly).
- (5.5) The inputs to spare gates are only basic events. Basic events that are inputs to spare gates can not be inputs for other types of gates (but they can be inputs to constraints).
- (5.6) Sequence enforcers must operate over the events of the fault tree.
- (5.7) Functional dependencies must be triggered by some event in the fault tree, every gate and basic event has a replication, the trigger must have a replication of 1, and only basic events can be dependent inputs.
- (5.8) All gates must have a replication of 1.

Note that full connectivity is not required by our specification—although all gates must have 1 or more inputs and must be input to the system level event, some of the basic events in the fault tree may not be inputs to any gate. This generality does not affect the semantics of dynamic fault trees, and will be useful in later specifications that build upon this one.

6 Failure Automata

In this section we specify the domain of failure automata.

6.1 States of Events and Histories

$$StateOfEvents == Event \rightarrow \mathbb{N}$$

StateOfEvents represents the state of all the events for a fault tree. Note that this does not capture the entire state of the fault tree; in particular, the allocation of spares to spare gates is not modeled.

$$History == \{ h : \mathrm{iseq}\ StateOfEvents \mid h \neq \langle\rangle \land (\forall i,j : \mathrm{dom}\, h \bullet \mathrm{dom}(h\,i) = \mathrm{dom}(h\,j)) \bullet h \}$$

A *History* is specified as a non-repeating sequence of *StateOfEvents* that represents the changing state of the fault tree over a set of discrete time steps. Every step in the history has the same set of events, although the event states can change.

6.2 Failure Automaton State

$$SpareInUse == \{ siu : Event \nrightarrow Event \mid siu \in \mathbb{F}(Event \times Event) \bullet siu \}$$

We declare *SparesInUse* as a finite partial function from *Event* to *Event*. The domain represents a subset of the spare gates in the fault tree, and the range is the spare being used by the spare gate (if any).

FailureAutomatonState

$$stateOfEvents : StateOfEvents$$
$$history : History$$
$$spareInUse : SpareInUse$$
$$systemFailedUncovered : Boolean$$

$$stateOfEvents = last\, history$$

The state of a fault tree consists of the state of the events, the history, the spare allocation, and the uncovered failure status. The *stateOfEvents* must be equal to the last state of events in the history.

6.3 Failure Automaton Transitions

FailureAutomatonTransition

$$from : FailureAutomatonState$$
$$to : FailureAutomatonState$$
$$causalBasicEvent : Event$$

$$to.history = from.history \frown \langle to.stateOfEvents \rangle$$
$$causalBasicEvent \in \mathrm{dom}\, from.stateOfEvents$$
$$from.stateOfEvents\, causalBasicEvent < to.stateOfEvents\, causalBasicEvent$$

A transition between states consists of a from state, a to state, and an associated causal basic event. The destination state extends the history of the source state by one set of event states. The causal basic event

must be one of the events in the event state, and it must be the case that additional replicate failures of the basic event occur in the transition between states.

6.4 Failure Automata

In this section we provide the complete specification of a failure automaton.

$$
\begin{array}{|l}
\textit{FailureAutomaton} \\
\hline
states : \mathbb{F}\, \textit{FailureAutomatonState} \\
transitions : \mathbb{F}\, \textit{FailureAutomatonTransition} \\
\hline
states = \bigcup \{ t : transitions \bullet \{ t.from, t.to \} \} \\
\hline
\end{array}
$$

A failure automaton consists of a finite set of states and transitions. The predicate constrains the transitions to be between the states of the failure automaton.

7 Semantics of Fault Trees in Terms of Failure Automata

In this section we specify the semantics of dynamic fault trees in terms of failure automata by establishing a correspondence between the two domains.

7.1 Number of Replicates, Event Occurrences

In this section we define functions for determining the number of replicates of an event that have occurred, the number of inputs to a gate that have occurred, and the notion of "in order" occurrence of inputs.

$$
\begin{array}{|l}
\mathit{NumberOfOccurredReplicatesInInputs} : \mathit{InputSequence} \times \mathit{StateOfEvents} \pfun \mathbb{N} \\
\hline
\mathrm{dom}\, \mathit{NumberOfOccurredReplicatesInInputs} = \\
\quad \{ \mathit{is} : \mathit{InputSequence};\ \mathit{soe} : \mathit{StateOfEvents} \mid \mathrm{ran}\, \mathit{is} \subseteq \mathrm{dom}\, \mathit{soe} \bullet (\mathit{is}, \mathit{soe}) \} \\
\forall \mathit{is} : \mathit{InputSequence};\ \mathit{soe} : \mathit{StateOfEvents} \mid \\
\quad (\mathit{is}, \mathit{soe}) \in \mathrm{dom}\, \mathit{NumberOfOccurredReplicatesInInputs} \bullet \\
\quad \mathit{NumberOfOccurredReplicatesInInputs}(\mathit{is}, \mathit{soe}) = \\
\qquad \textbf{if } \mathit{is} = \langle\rangle \textbf{ then } 0 \\
\qquad \textbf{else } \mathit{soe}(\mathit{head}\, \mathit{is}) + \mathit{NumberOfOccurredReplicatesInInputs}((\mathit{tail}\, \mathit{is}), \mathit{soe})
\end{array}
$$

Given a sequence of inputs and a set of event states, this function determines the number of replicates that have occurred for all the input events.

$$
\begin{array}{|l}
\mathit{OccursInTimeStep}\_ : \mathbb{P}(\mathit{History} \times \mathit{Event} \times \mathbb{N}_1) \\
\hline
\forall h : \mathit{History};\ e : \mathit{Event};\ t : \mathbb{N}_1 \mid t \leq \# h \wedge e \in \mathrm{dom}(h\, 1) \bullet \\
\quad \mathit{OccursInTimeStep}(h, e, t) \Leftrightarrow \\
\qquad t = 1 \wedge h\, t\, e > 0 \vee t > 1 \wedge h\, t\, e > h\, (t-1)\, e
\end{array}
$$

This function determines whether an event had a replicate that occurred in a given time step. We allow events to occur in the initial state. (In fact, spare contention in the initial state can cause multiple nondeterministic initial states.)

$$
\begin{array}{|l}
\mathit{FirstOccurrenceTime} : \mathit{History} \times \mathit{Event} \pfun \mathbb{N} \\
\hline
\mathrm{dom}\, \mathit{FirstOccurrenceTime} = \{ h : \mathit{History};\ e : \mathit{Event} \mid e \in \mathrm{dom}(h\, 1) \bullet (h, e) \} \\
\forall h : \mathit{History};\ e : \mathit{Event};\ t : \mathbb{N}_1 \mid (h, e) \in \mathrm{dom}\, \mathit{FirstOccurrenceTime} \bullet \\
\quad \mathit{FirstOccurrenceTime}(h, e) = \\
\qquad \textbf{if } \mathit{OccursInTimeStep}(h, e, t) \wedge \\
\qquad\quad (\forall t_2 : \mathbb{N}_1 \mid t_2 < t \bullet \neg\, \mathit{OccursInTimeStep}(h, e, t_2)) \\
\qquad \textbf{then } t \textbf{ else } 0
\end{array}
$$

This function determines the time step in which the first replicate of an event occurs. A value of 0 indicates that no replicate has failed.

$$FirstFullOccurrenceTime : History \times Event \times \mathbb{N} \pfun \mathbb{N}$$

$$\begin{aligned}
&\dom FirstFullOccurrenceTime = \\
&\quad \{ h : History;\ e : Event;\ r : \mathbb{N} \mid e \in \dom(h\,1) \bullet (h,e,r) \} \\
&\forall h : History;\ e : Event;\ r : \mathbb{N};\ t : \mathbb{N}_1 \mid \\
&\quad t \in \dom h \land (h,e,r) \in \dom FirstFullOccurrenceTime \bullet \\
&\quad FirstFullOccurrenceTime(h,e,r) = \\
&\qquad \textbf{if } h\,t\,e = r \land OccursInTimeStep(h,e,t) \textbf{ then } t \textbf{ else } 0
\end{aligned}$$

This function computes the history position in which the final replicate of an event occurs. It returns 0 if the first replicate of an event has not occurred (i.e. the number operational is greater than 0 throughout the history).

$$InputsOccurredAndInOrder\_ : \mathbb{P}(InputSequence \times History \times ReplicationMap)$$

$$\begin{aligned}
&\forall is : InputSequence;\ h : History;\ rs : ReplicationMap \mid \ran is \subseteq \dom(h\,1) \bullet \\
&\quad InputsOccurredAndInOrder(is,h,rs) \Leftrightarrow \\
&\qquad NumberOfOccurredReplicatesInInputs(is,last\,h) = \qquad (7.1) \\
&\qquad\quad NumberOfReplicatesInInputs(is,rs) \land \\
&\qquad (\forall i,j : \dom is \mid i < j \bullet \qquad (7.2) \\
&\qquad\quad FirstFullOccurrenceTime(h,is\,i,rs(is\,i)) \neq 0 \land \\
&\qquad\quad FirstOccurrenceTime(h,is\,j) \neq 0 \land \\
&\qquad\quad FirstFullOccurrenceTime(h,is\,i,rs(is\,i)) \leq \\
&\qquad\qquad FirstOccurrenceTime(h,is\,j))
\end{aligned}$$

Given a history and a sequence of events, the value of the *InputsOccurredAndInOrder* function is true if the replicates in each position fail before or at the same time as the replicates in later positions. Predicate 7.1 states that all the inputs must be fully occurred, and predicate 7.2 states that the event at position i must be fully occurred at or before the time at which the first replicate at position $i+1$ occurs.

$$NewlyOccurredBasicEvents : FaultTree \times History \pfun \mathbb{F}\, Event$$

$$\begin{aligned}
&\dom NewlyOccurredBasicEvents = \\
&\quad \{ ft : FaultTree;\ h : History \mid ft.events = \dom(h\,1) \bullet (ft,h) \} \\
&\forall ft : FaultTree;\ h : History \mid (ft,h) \in \dom NewlyOccurredBasicEvents \bullet \\
&\quad NewlyOccurredBasicEvents(ft,h) = \\
&\qquad \{ e : Event \mid e \in ft.basicEvents \land \\
&\qquad\quad e \in \dom(last\,h) \land OccursInTimeStep(h,e,\#h) \bullet e \}
\end{aligned}$$

This helper function computes the set of basic events that occurred in the last history step.

7.2 Semantics of AND Gates

$$FaultTreeAndFailureAutomatonEventsMatch\_ : \mathbb{P}(FaultTree \times FailureAutomaton)$$

$$\begin{aligned}
&\forall ft : FaultTree;\ fa : FailureAutomaton \bullet \\
&\quad FaultTreeAndFailureAutomatonEventsMatch(ft,fa) \Leftrightarrow \\
&\qquad (\forall fas : FailureAutomatonState \mid fas \in fa.states \bullet \\
&\qquad ft.events = \dom fas.stateOfEvents)
\end{aligned}$$

In order to ensure the correct behavior of events in a fault tree with respect to the failure automaton, it

must be the case that the events in the fault tree have corresponding states in the failure automaton. This function ensures that this condition is satisfied.

$$
\begin{array}{|l}
ANDSemantics\_ : \mathbb{P}(FaultTree \times FailureAutomaton) \\
\hline
\forall ft : FaultTree; fa : FailureAutomaton \mid \\
\quad FaultTreeAndFailureAutomatonEventsMatch(ft,fa) \bullet \\
\quad ANDSemantics(ft,fa) \Leftrightarrow \\
\qquad (\forall g : ft.andGates; fas : FailureAutomatonState \mid fas \in fa.states \bullet \\
\qquad\quad NumberOfOccurredReplicatesInInputs(ft.inputs\, g, fas.stateOfEvents) = \\
\qquad\qquad NumberOfReplicatesInInputs(ft.inputs\, g, ft.replications) \Rightarrow \\
\qquad\qquad fas.stateOfEvents\, g = 1 \wedge \\
\qquad\quad NumberOfOccurredReplicatesInInputs(ft.inputs\, g, fas.stateOfEvents) \neq \\
\qquad\qquad NumberOfReplicatesInInputs(ft.inputs\, g, ft.replications) \Rightarrow \\
\qquad\qquad fas.stateOfEvents\, g = 0)
\end{array}
$$

Every AND gate in the fault tree is an event whose individual state in all of the failure automaton states is defined by this schema. The first predicate specifies that the event associated with the gate occurs if all the inputs have occurred (i.e. the number of occurred input replicates is equal to the total number of input replicates). The second predicate specifies that the event associated with the gate does not occur if it is not the case that all the input replicates have occurred.

7.3 Semantics of OR Gates

$$
\begin{array}{|l}
ORSemantics\_ : \mathbb{P}(FaultTree \times FailureAutomaton) \\
\hline
\forall ft : FaultTree; fa : FailureAutomaton \mid \\
\quad FaultTreeAndFailureAutomatonEventsMatch(ft,fa) \bullet \\
\quad ORSemantics(ft,fa) \Leftrightarrow \\
\qquad (\forall g : ft.orGates; fas : FailureAutomatonState \mid fas \in fa.states \bullet \\
\qquad\quad NumberOfOccurredReplicatesInInputs(ft.inputs\, g, fas.stateOfEvents) \neq 0 \\
\qquad\qquad \Rightarrow fas.stateOfEvents\, g = 1 \wedge \\
\qquad\quad NumberOfOccurredReplicatesInInputs(ft.inputs\, g, fas.stateOfEvents) = 0 \\
\qquad\qquad \Rightarrow fas.stateOfEvents\, g = 0)
\end{array}
$$

This schema defines the semantics of all of the OR gates in a fault tree. The OR gate's associated event occurs if any of the input events have occurred. Otherwise, the associated event does not occur.

7.4 Semantics of Threshold Gates

$$
\begin{array}{|l}
ThresholdSemantics\_ : \mathbb{P}(FaultTree \times FailureAutomaton) \\
\hline
\forall ft : FaultTree; fa : FailureAutomaton \mid \\
\quad FaultTreeAndFailureAutomatonEventsMatch(ft,fa) \bullet \\
\quad ThresholdSemantics(ft,fa) \Leftrightarrow \\
\qquad (\forall g : ft.thresholdGates; fas : FailureAutomatonState \mid fas \in fa.states \bullet \\
\qquad\quad NumberOfOccurredReplicatesInInputs(ft.inputs\, g, fas.stateOfEvents) \geq \\
\qquad\qquad ft.thresholds\, g \Rightarrow fas.stateOfEvents\, g = 1 \wedge \\
\qquad\quad NumberOfOccurredReplicatesInInputs(ft.inputs\, g, fas.stateOfEvents) < \\
\qquad\qquad ft.thresholds\, g \Rightarrow fas.stateOfEvents\, g = 0)
\end{array}
$$

Like the specifications for the AND and OR gates, the threshold gate specification is based on the number of input replicates that have occurred. In this case, the event associated with the threshold gate occurs only if the number of occurred inputs is greater than or equal to the threshold gate's k value.

7.5 Semantics of PAND Gates

$$
\begin{array}{|l}
PANDSemantics\_ : \mathbb{P}(FaultTree \times FailureAutomaton) \\
\hline
\forall ft : FaultTree; fa : FailureAutomaton \mid \\
\quad FaultTreeAndFailureAutomatonEventsMatch(ft,fa) \bullet \\
\quad PANDSemantics(ft,fa) \Leftrightarrow \\
\qquad (\forall g : ft.pandGates; fas : FailureAutomatonState \mid fas \in fa.states \bullet \\
\qquad\quad InputsOccurredAndInOrder(ft.inputs\, g, fas.history, ft.replications) \\
\qquad\qquad \Rightarrow fas.stateOfEvents\, g = 1 \land \\
\qquad\quad \neg\, InputsOccurredAndInOrder(ft.inputs\, g, fas.history, ft.replications) \\
\qquad\qquad \Rightarrow fas.stateOfEvents\, g = 0)
\end{array}
$$

A PAND gate's event occurs if all the inputs have occurred in order, and does not occur otherwise.

7.6 Semantics of Spare Gates

In this section we present the specification of spare gates. Spare gates are easily the most semantically rich construct in the DFT modeling language. We will present the specification in two parts in order to simplify the complexity: the state semantics which describe allocation of spares for a spare gate, and the transition semantics which describe the reallocation of spares resulting from the occurrence of a basic event.

$$
\begin{array}{|l}
NumberOfSpareGatesUsingSpare : SpareInUse \times Event \rightarrow \mathbb{N} \\
\hline
\forall siu : SpareInUse; e : Event \bullet \\
\quad NumberOfSpareGatesUsingSpare(siu, e) = \#(siu \rhd \{e\})
\end{array}
$$

NumberOfSpareGatesUsingSpare determines the number of spare gates that have a particular event allocated to them. The predicate states that the value of the function is defined as the size of the result of restricting the spares in use relation to only those spare gates that use event e.

$$
\begin{array}{|l}
FaultTreeAndFailureAutomatonSGsMatch\_ : \mathbb{P}(FaultTree \times FailureAutomaton) \\
\hline
\forall ft : FaultTree; fa : FailureAutomaton \bullet \\
\quad FaultTreeAndFailureAutomatonSGsMatch(ft,fa) \Leftrightarrow \\
\qquad (\forall fas : FailureAutomatonState \mid fas \in fa.states \bullet \\
\qquad \mathrm{dom} fas.spareInUse \subseteq ft.spareGates \land \mathrm{ran} fas.spareInUse \subseteq ft.basicEvents)
\end{array}
$$

In order to ensure the correct behavior of spare gates in a fault tree with respect to the failure automaton, it must be the case that the events referenced in the spare in use relation must correspond to events in the fault tree. This function ensures that this condition is satisfied.

$$
\begin{array}{|l}
SpareInUseIsAnInput\_ : \mathbb{P}(FaultTree \times FailureAutomaton) \\
\hline
\forall ft : FaultTree; fa : FailureAutomaton \mid \\
\quad FaultTreeAndFailureAutomatonEventsMatch(ft,fa) \land \\
\quad FaultTreeAndFailureAutomatonSGsMatch(ft,fa) \bullet \\
\quad\quad SpareInUseIsAnInput(ft,fa) \Leftrightarrow \\
\quad\quad\quad (\forall fas : FailureAutomatonState; sgus : Event \mid \\
\quad\quad\quad fas \in fa.states \land sgus \in \mathrm{dom} fas.spareInUse \bullet \\
\quad\quad\quad fas.spareInUse\, sgus \in \mathrm{ran}(ft.inputs\, sgus))
\end{array}
$$

This function ensures that any spare being used is an input to the spare gate using it.

$$
\begin{array}{|l}
ReplicatesNotOverused\_ : \mathbb{P}(FaultTree \times FailureAutomaton) \\
\hline
\forall ft : FaultTree; fa : FailureAutomaton \mid \\
\quad FaultTreeAndFailureAutomatonEventsMatch(ft,fa) \land \\
\quad FaultTreeAndFailureAutomatonSGsMatch(ft,fa) \bullet \\
\quad\quad ReplicatesNotOverused(ft,fa) \Leftrightarrow \\
\quad\quad\quad (\forall fas : FailureAutomatonState; sp : Event \mid \\
\quad\quad\quad fas \in fa.states \land sp \in \mathrm{ran} fas.spareInUse \bullet \\
\quad\quad\quad NumberOfSpareGatesUsingSpare(fas.spareInUse, sp) \leq \\
\quad\quad\quad\quad ft.replications\, sp - fas.stateOfEvents\, sp)
\end{array}
$$

This function ensures that the number of spare gates using a replicate of a spare never exceeds the number of available replicates of the spare.

$$
\begin{array}{|l}
PreviousSparesUnavailable\_ : \mathbb{P}(FaultTree \times FailureAutomaton) \\
\hline
\forall ft : FaultTree; fa : FailureAutomaton \mid \\
\quad FaultTreeAndFailureAutomatonEventsMatch(ft,fa) \land \\
\quad FaultTreeAndFailureAutomatonSGsMatch(ft,fa) \bullet \\
\quad\quad PreviousSparesUnavailable(ft,fa) \Leftrightarrow \\
\quad\quad\quad (\forall fas : FailureAutomatonState; sgus : Event; i : \mathbb{N}_1 \mid \\
\quad\quad\quad fas \in fa.states \land sgus \in \mathrm{dom} fas.spareInUse \land \\
\quad\quad\quad i \in \mathrm{dom}(ft.inputs\, sgus) \land fas.spareInUse\, sgus = ft.inputs\, sgus\, i \bullet \\
\quad\quad\quad\quad (\forall j : 1 \,..\, i-1; be : Event \mid be = ft.inputs\, sgus\, j \bullet \\
\quad\quad\quad\quad NumberOfSpareGatesUsingSpare(fas.spareInUse, be) = \\
\quad\quad\quad\quad\quad ft.replications\, be - fas.stateOfEvents\, be))
\end{array}
$$

This function ensures that if a spare is being used, all the previous spares in the spare gate's sequence of inputs have no available spare replicates.

$$
\begin{array}{|l}
NoSparesInUseOnlyIfNoneAvailable\_ : \mathbb{P}(FaultTree \times FailureAutomaton) \\
\hline
\forall ft : FaultTree; fa : FailureAutomaton \mid \\
\quad FaultTreeAndFailureAutomatonEventsMatch(ft,fa) \land \\
\quad FaultTreeAndFailureAutomatonSGsMatch(ft,fa) \bullet \\
\qquad NoSparesInUseOnlyIfNoneAvailable(ft,fa) \Leftrightarrow \\
\qquad\quad (\forall fas : FailureAutomatonState; sgnus : Event \mid \\
\qquad\quad fas \in fa.states \land sgnus \notin \mathrm{dom} fas.spareInUse \land sgnus \in \mathrm{dom} ft.inputs \bullet \\
\qquad\qquad (\forall be : Event \mid be \in \mathrm{ran}(ft.inputs\, sgnus) \bullet \\
\qquad\qquad NumberOfSpareGatesUsingSpare(fas.spareInUse, be) = \\
\qquad\qquad\quad ft.replications\, be - fas.stateOfEvents\, be))
\end{array}
$$

This function ensures that a spare gate uses a spare if one is available.

$$
\begin{array}{|l}
SpareGateStateSemantics\_ : \mathbb{P}(FaultTree \times FailureAutomaton) \\
\hline
\forall ft : FaultTree; fa : FailureAutomaton \mid \\
\quad FaultTreeAndFailureAutomatonEventsMatch(ft,fa) \land \\
\quad FaultTreeAndFailureAutomatonSGsMatch(ft,fa) \bullet \\
\quad SpareGateStateSemantics(ft,fa) \Leftrightarrow \\
\qquad SpareInUseIsAnInput(ft,fa) \land \\
\qquad ReplicatesNotOverused(ft,fa) \land \\
\qquad PreviousSparesUnavailable(ft,fa) \land \\
\qquad NoSparesInUseOnlyIfNoneAvailable(ft,fa) \land \\
\qquad (\forall sg : Event; fas : FailureAutomatonState \mid \\
\qquad\quad sg \in ft.spareGates \land fas \in fa.states \bullet \\
\qquad\quad (sg \in \mathrm{dom} fas.spareInUse \Rightarrow fas.stateOfEvents\, sg = 0) \land \\
\qquad\quad (sg \notin \mathrm{dom} fas.spareInUse \Rightarrow fas.stateOfEvents\, sg = 1))
\end{array}
$$

$$
\begin{array}{|l}
IsUsingSpare\_ : \mathbb{P}(Event \times FailureAutomatonState) \\
\hline
\forall sg : Event; fas : FailureAutomatonState \bullet \\
\quad IsUsingSpare(sg,fas) \Leftrightarrow sg \in \mathrm{dom} fas.spareInUse
\end{array}
$$

This helper function determines whether a spare gate is using a spare.

$$
\begin{array}{|l}
SpareBeingUsed : Event \times FailureAutomatonState \pfun Event \\
\hline
\mathrm{dom}\, SpareBeingUsed = \\
\quad \{ sg : Event; fas : FailureAutomatonState \mid IsUsingSpare(sg,fas) \bullet (sg,fas) \} \\
\forall sg : Event; fas : FailureAutomatonState \mid (sg,fas) \in \mathrm{dom}\, SpareBeingUsed \bullet \\
\quad SpareBeingUsed(sg,fas) = fas.spareInUse\, sg
\end{array}
$$

This is a helper function that computes the spare that a spare gate is using in a given fault tree.

$$
\begin{array}{|l}
SpareGateStaysFailed\_ : \mathbb{P}(FaultTree \times FailureAutomaton) \\
\hline
\forall ft : FaultTree; fa : FailureAutomaton \mid \\
\quad FaultTreeAndFailureAutomatonEventsMatch(ft,fa) \land \\
\quad FaultTreeAndFailureAutomatonSGsMatch(ft,fa) \bullet \\
\quad SpareGateStaysFailed(ft,fa) \Leftrightarrow \\
\qquad (\forall sg : Event; fat : FailureAutomatonTransition \mid \\
\qquad sg \in ft.spareGates \land fat \in fa.transitions \bullet \\
\qquad\quad \neg\, IsUsingSpare(sg,fat.from) \Rightarrow \neg\, IsUsingSpare(sg,fat.to))
\end{array}
$$

This function ensures that a spare gate that is not using a spare (is failed) remains so.

$$
\begin{array}{|l}
SpareGateContinuesUsingOperationalSpare\_ : \mathbb{P}(FaultTree \times FailureAutomaton) \\
\hline
\forall ft : FaultTree; fa : FailureAutomaton \mid \\
\quad FaultTreeAndFailureAutomatonEventsMatch(ft,fa) \land \\
\quad FaultTreeAndFailureAutomatonSGsMatch(ft,fa) \bullet \\
\quad SpareGateContinuesUsingOperationalSpare(ft,fa) \Leftrightarrow \\
\qquad (\forall sg : Event; fat : FailureAutomatonTransition \mid \\
\qquad sg \in ft.spareGates \land fat \in fa.transitions \bullet \\
\qquad\quad IsUsingSpare(sg,fat.from) \land SpareBeingUsed(sg,fat.from) \notin \\
\qquad\qquad NewlyOccurredBasicEvents(ft,fat.to.history) \Rightarrow \\
\qquad\quad IsUsingSpare(sg,fat.to) \land \\
\qquad\qquad SpareBeingUsed(sg,fat.to) = SpareBeingUsed(sg,fat.from))
\end{array}
$$

This function ensures that a spare gate that is using a spare which has not failed continues to use the same spare.

$$
\begin{array}{|l}
SpareGateUsesAvailableSpare\_ : \mathbb{P}(FaultTree \times FailureAutomaton) \\
\hline
\forall ft : FaultTree; fa : FailureAutomaton \mid \\
\quad FaultTreeAndFailureAutomatonEventsMatch(ft,fa) \land \\
\quad FaultTreeAndFailureAutomatonSGsMatch(ft,fa) \bullet \\
\quad SpareGateUsesAvailableSpare(ft,fa) \Leftrightarrow \\
\qquad (\forall sg : Event; fat : FailureAutomatonTransition \mid \\
\qquad sg \in ft.spareGates \land fat \in fa.transitions \bullet \\
\qquad\quad IsUsingSpare(sg,fat.from) \land SpareBeingUsed(sg,fat.from) \in \\
\qquad\quad NewlyOccurredBasicEvents(ft,fat.to.history) \Rightarrow \\
\qquad\qquad \neg\, IsUsingSpare(sg,fat.to) \lor \\
\qquad\qquad IsUsingSpare(sg,fat.to) \land \\
\qquad\qquad (\exists i,j : \mathbb{N}_1 \mid i \in \mathrm{dom}(ft.inputs\, sg) \land j \in \mathrm{dom}(ft.inputs\, sg) \land \\
\qquad\qquad SpareBeingUsed(sg,fat.from) = ft.inputs\, sg\, i \land \\
\qquad\qquad SpareBeingUsed(sg,fat.to) = ft.inputs\, sg\, j \bullet i \leq j))
\end{array}
$$

If a spare is being used in the first fault tree, then either no spare is used in the second, or a later spare is used.

$$
\begin{array}{|l}
SpareGateTransitionSemantics\_ : \mathbb{P}(FaultTree \times FailureAutomaton) \\
\hline
\forall ft : FaultTree; fa : FailureAutomaton \mid \\
\quad FaultTreeAndFailureAutomatonEventsMatch(ft,fa) \land \\
\quad FaultTreeAndFailureAutomatonSGsMatch(ft,fa) \bullet \\
\quad SpareGateTransitionSemantics(ft,fa) \Leftrightarrow \\
\qquad (\forall sg : ft.spareGates; fat : FailureAutomatonTransition \mid \\
\qquad sg \in ft.spareGates \land fat \in fa.transitions \bullet \\
\qquad\quad SpareGateStaysFailed(ft,fa) \land \\
\qquad\quad SpareGateContinuesUsingOperationalSpare(ft,fa) \land \\
\qquad\quad SpareGateUsesAvailableSpare(ft,fa))
\end{array}
$$

This schema specifies the semantics of sparing as the state of the fault tree changes. It ensures that the spare allocation is consistent across a state transition from the *fat.from* state of a transition *fat* to the *fat.to* state.

$$
\begin{array}{|l}
SpareGateSemantics\_ : \mathbb{P}(FaultTree \times FailureAutomaton) \\
\hline
\forall ft : FaultTree; fa : FailureAutomaton \mid \\
\quad FaultTreeAndFailureAutomatonEventsMatch(ft,fa) \land \\
\quad FaultTreeAndFailureAutomatonSGsMatch(ft,fa) \bullet \\
\quad SpareGateSemantics(ft,fa) \Leftrightarrow \\
\qquad SpareGateStateSemantics(ft,fa) \land SpareGateTransitionSemantics(ft,fa)
\end{array}
$$

The complete spare gate semantics is the conjunction of the state and transition semantics.

7.7 Semantics of Sequence Enforcing Constraints

$$
\begin{array}{|l}
SEQSemantics\_ : \mathbb{P}(FaultTree \times FailureAutomaton) \\
\hline
\forall ft : FaultTree; fa : FailureAutomaton \mid \\
\quad FaultTreeAndFailureAutomatonEventsMatch(ft,fa) \bullet \\
\quad SEQSemantics(ft,fa) \Leftrightarrow \\
\qquad (\forall seq : InputSequence; fas : FailureAutomatonState \mid \\
\qquad seq \in ft.seqs \land fas \in fa.states \bullet \\
\qquad\quad \forall i : 1 \mathinner{..} \#seq \mid fas.stateOfEvents(seq\, i) > 0 \bullet \\
\qquad\qquad (\forall j : 1 \mathinner{..} i-1 \bullet \\
\qquad\qquad\quad fas.stateOfEvents(seq\, j) = \\
\qquad\qquad\qquad ft.replications(seq\, j) \land \\
\qquad\qquad\qquad FirstFullOccurrenceTime(fas.history, seq\, j, ft.replications(seq\, j)) \\
\qquad\qquad\qquad\quad \leq FirstOccurrenceTime(fas.history, seq\, i)))
\end{array}
$$

A sequence enforcing gate disallows certain sequences of events from occurring. The SEQ gate has the same ordering semantics as the PAND gate with respect to simultaneous occurrence and replicated inputs.

7.8 Semantics of Functional Dependency Constraints

$$
\begin{array}{|l}
FDEPSemantics\_ : \mathbb{P}(FaultTree \times FailureAutomaton) \\
\hline
\forall ft : FaultTree;\ fa : FailureAutomaton \mid \\
\quad FaultTreeAndFailureAutomatonEventsMatch(ft,fa) \bullet \\
\quad FDEPSemantics(ft,fa) \Leftrightarrow \\
\qquad (\forall tr : Event;\ ds : InputSequence;\ fas : FailureAutomatonState;\ t : \mathbb{N}_1 \mid \\
\qquad fas \in fa.states \land (tr,ds) \in ft.fdeps \land 1 \leq t \leq \#(fas.history) \bullet \\
\qquad\quad fas.stateOfEvents\, tr = 1 \Rightarrow \\
\qquad\qquad NumberOfOccurredReplicatesInInputs(ds,(fas.history\, t)) = \\
\qquad\qquad NumberOfReplicatesInInputs(ds,ft.replications))
\end{array}
$$

For a given history, if the trigger event occurs in a time step of the history, then all the replicates of all the dependent events (ds) also occur.

7.9 Uncovered Failure Semantics

$$
\begin{array}{|l}
UncoveredFailureSemantics\_ : \mathbb{P}(FailureAutomaton) \\
\hline
\forall fa : FailureAutomaton \bullet \\
\quad UncoveredFailureSemantics(fa) \Leftrightarrow \\
\qquad (\forall fat : FailureAutomatonTransition \bullet \\
\qquad\quad fat.from.systemFailedUncovered = True \Rightarrow \\
\qquad\qquad fat.to.systemFailedUncovered = True)
\end{array}
$$

This function ensures that failure automaton states which are failed uncovered remain failed uncovered.

7.10 Causal Basic Event Semantics

In this section we specify the relationship between the causal basic event and the change in fault tree state.

$$
\begin{array}{|l}
CausalBasicEventSemantics\_ : \mathbb{P}(FaultTree \times FailureAutomaton) \\
\hline
\forall ft : FaultTree;\ fa : FailureAutomaton \mid \\
\quad FaultTreeAndFailureAutomatonEventsMatch(ft,fa) \bullet \\
\quad CausalBasicEventSemantics(ft,fa) \Leftrightarrow \\
\qquad (\forall fat : fa.transitions \bullet \\
\qquad\quad fat.causalBasicEvent \in ft.basicEvents) \land \\
\qquad (\forall fas : FailureAutomatonState;\ be : Event \mid \\
\qquad\quad fas \in fa.states \land be \in ft.basicEvents \bullet \\
\qquad\qquad (fas.stateOfEvents\, be < ft.replications\, be \Rightarrow \\
\qquad\qquad (\exists fat : fa.transitions \bullet fat.from = fas \land fat.causalBasicEvent = be)))
\end{array}
\tag{7.3}
$$

In this function, we ensure that the causal basic event of a failure automaton transition is a basic event of the fault tree, and that every basic event that can fail in a given state does.

7.11 Complete Fault Tree Semantics in terms of Failure Automata

$$
\begin{array}{|l}
FaultTreeSemantics : FaultTree \rightarrow FailureAutomaton \\
\hline
\forall ft : FaultTree; fa : FailureAutomaton \bullet \\
\quad FaultTreeSemantics(ft) = fa \Leftrightarrow \\
\qquad FaultTreeAndFailureAutomatonEventsMatch(ft,fa) \wedge \\
\qquad FaultTreeAndFailureAutomatonSGsMatch(ft,fa) \wedge \\
\qquad CausalBasicEventSemantics(ft,fa) \wedge UncoveredFailureSemantics(fa) \wedge \\
\qquad ANDSemantics(ft,fa) \wedge ORSemantics(ft,fa) \wedge \\
\qquad ThresholdSemantics(ft,fa) \wedge PANDSemantics(ft,fa) \wedge \\
\qquad SpareGateSemantics(ft,fa) \wedge SEQSemantics(ft,fa) \wedge \\
\qquad FDEPSemantics(ft,fa)
\end{array}
$$

The complete fault tree semantics, expressed in terms of failure automata, is the conjunction of the gate, constraint, causal event, and system failure semantics.

8 Markov Models

In this section we provide an abstract definition of Markov models.

$[MarkovModelStateID]$

A Markov state identifier is used to distinguish Markov states.

$$
\begin{array}{|l}
\textit{MarkovModelState} \\
\hline
id : MarkovModelStateID \\
initialProbability : Probability \\
finalProbability : Probability \\
\hline
\end{array}
$$

A Markov state has an associated initial and final state probability.

$$TransitionRateFunction == Time \rightarrow Rate$$

A transition rate function is a function of time that computes a rate.

$$
\begin{array}{|l}
\textit{MarkovModelTransition} \\
\hline
from : MarkovModelState \\
to : MarkovModelState \\
transitionRateFunction : TransitionRateFunction \\
\hline
\end{array}
$$

A Markov transition consists of a from and to state, and a transition rate function.

$$
\begin{array}{|l}
\textit{MarkovModel} \\
\hline
states : \mathbb{F}\, MarkovModelState \\
transitions : \mathbb{F}\, MarkovModelTransition \\
\hline
states = \bigcup \{ t : transitions \bullet \{ t.from, t.to \} \} \\
\exists is : \mathbb{F}\, MarkovModelState \mid \\
\quad is = \{ st : states \mid \neg\, (\exists tr : MarkovModelTransition \mid \\
\qquad tr \in transitions \bullet st = tr.to) \bullet st \} \land \\
\quad \#is > 0 \bullet \\
\quad \forall st : states \bullet st.initialProbability = \\
\qquad \textbf{if } st \in is \textbf{ then } 1_{\mathbb{R}} /_{\mathbb{R}} intToReal(\#is) \textbf{ else } 0_{\mathbb{R}} \\
\hline
\end{array}
$$

A Markov model comprises a set of states and a set of transitions between states. The first predicate states that the transitions must be over the particular set of states *states*. The second predicate assigns the state probability to 0 if the state is not an initial state, and otherwise distributes the overall probability of 1 across the initial states. The semantics of the Markov model are incomplete—we have not specified how the final state probabilities are computed from the transition rates and the initial state probabilities. However, Markov models are well enough understood that we are willing to elide the details.

9 Basic Event Models

We now shift our attention to one of the parameters of the semantics of FAs in terms of MMs–the basic event models associated with the basic events of the fault tree.

$$
\begin{array}{|l}
\textit{CoverageModel} \\
\hline
\textit{singlePointFailureProbability} : \textit{Probability} \\
\textit{coveredFailureProbability} : \textit{Probability} \\
\textit{restorationProbability} : \textit{Probability} \\
\hline
\textit{singlePointFailureProbability} +_{\mathbb{R}} \textit{coveredFailureProbability} +_{\mathbb{R}} \\
\quad \textit{restorationProbability} = 1_{\mathbb{R}} \\
\hline
\end{array}
$$

The *restorationProbability* is the probability that the component masks an internal failure. The *coveredFailureProbability* is the probability that a component fails in a way that can be detected by the system. The *singlePointFailureProbability* is the probability that the component fails and brings down the system. The sum of these three parameters must be one.

$$[\textit{Distribution}]$$

We introduce the general *Distribution* type, from which we will define subtypes with associated failure parameters.

$$
\begin{array}{|l}
\textit{constantDistributions} : \mathbb{F}\, \textit{Distribution} \\
\textit{probabilities} : \textit{Distribution} \nrightarrow \textit{Probability} \\
\hline
\mathrm{dom}\, \textit{probabilities} = \textit{constantDistributions} \\
\hline
\end{array}
$$

The *constantDistributions* are a subset of *Distribution*. Each constant distribution has an associated probability. The definition of this and other distributions is axiomatic.

$$
\begin{array}{|l}
\textit{exponentialDistributions} : \mathbb{F}\, \textit{Distribution} \\
\textit{exponentialRates} : \textit{Distribution} \nrightarrow \textit{Rate} \\
\hline
\mathrm{dom}\, \textit{exponentialRates} = \textit{exponentialDistributions} \\
\hline
\end{array}
$$

The *exponentialDistributions* are a subset of *Distribution*. Each exponential distribution has an associated rate.

$$\textit{WeibullShape} == \{ r : \mathbb{R} \mid r \geq_{\mathbb{R}} 0_{\mathbb{R}} \bullet r \}$$

The Weibull shape is greater than or equal to 0.

$$
\begin{array}{|l}
\textit{weibullDistributions} : \mathbb{F}\, \textit{Distribution} \\
\textit{weibullRates} : \textit{Distribution} \nrightarrow \textit{Rate} \\
\textit{weibullShapes} : \textit{Distribution} \nrightarrow \textit{WeibullShape} \\
\hline
\mathrm{dom}\, \textit{weibullRates} = \textit{weibullDistributions} \\
\mathrm{dom}\, \textit{weibullShapes} = \textit{weibullDistributions} \\
\hline
\end{array}
$$

The *weibullDistributions* are a subset of *Distribution*. Each Weibull distribution has an associated rate

and shape.

$$LogNormalMean == \{ r : \mathbb{R} \mid r \geq_{\mathbb{R}} 0_{\mathbb{R}} \bullet r \}$$

$$LogNormalStdDev == \{ r : \mathbb{R} \mid r \geq_{\mathbb{R}} 0_{\mathbb{R}} \bullet r \}$$

The lognormal mean and standard distribution must be greater than or equal to 0.

$$\begin{array}{|l} logNormalDistributions : \mathbb{F}\, Distribution \\ logNormalMeans : Distribution \pfun LogNormalMean \\ logNormalStdDevs : Distribution \pfun LogNormalStdDev \\ \hline \mathrm{dom}\, logNormalMeans = logNormalDistributions \\ \mathrm{dom}\, logNormalStdDevs = logNormalDistributions \end{array}$$

The *logNormalDistributions* are a subset of *Distribution*. Each lognormal distribution has an associated mean and standard deviation.

$$Dormancy == \{ r : \mathbb{R} \mid 0_{\mathbb{R}} \leq_{\mathbb{R}} r \geq_{\mathbb{R}} 1_{\mathbb{R}} \bullet r \}$$

$$\mathrm{disjoint}\ \langle constantDistributions, exponentialDistributions, weibullDistributions, logNormalDistributions \rangle$$

The various types of distributions are disjoint.

BasicEventModel

$$\begin{array}{|l} distribution : Distribution \\ coverageModel : CoverageModel \\ dormancy : Dormancy \end{array}$$

A *BasicEventModel* describes the stochastic behavior of a basic event. The dormancy must be between 0 and 1 inclusive.

$$BasicEventModelFunction == Event \pfun BasicEventModel$$

A basic event model function maps basic events to their associated models.

10 Semantics of Fault Tree Automata in Terms of Markov Chains

We now specify the semantics of failure automata in terms of Markov models.

10.1 Structural Correspondence Between Failure Automata and Markov Models

In this section we establish the structural correspondence between failure automata and Markov models.

$$StateCorrespondence == FailureAutomatonState \rightarrowtail\!\!\!\!\!\!\rightarrow MarkovModelState$$

There is a bijection between states in the failure automaton and the Markov model.

$$TransitionCorrespondence == FailureAutomatonTransition \rightarrow MarkovModelTransition$$

A transition correspondence is a mapping from failure automaton transitions to Markov model transitions. It is not a bijection because multiple FA transitions can map to the same MM transition.

$$\begin{array}{|l} StructuralCorrespondenceExists\_ : \mathbb{P}(FailureAutomaton \times MarkovModel) \\ \hline \forall fa : FailureAutomaton;\ mm : MarkovModel \bullet \\ \quad StructuralCorrespondenceExists(fa, mm) \Leftrightarrow \\ \qquad (\exists fas2mms : StateCorrespondence;\ fat2mmt : TransitionCorrespondence \bullet \\ \qquad\quad \mathrm{dom}\, fas2mms = fa.states \land \mathrm{ran}\, fas2mms = mm.states \land \\ \qquad\quad \mathrm{dom}\, fat2mmt = fa.transitions \land \mathrm{ran}\, fat2mmt = mm.transitions \land \\ \qquad\quad (\forall fat : FailureAutomatonTransition \mid fat \in fa.transitions \bullet \\ \qquad\qquad (fas2mms\, fat.from = (fat2mmt\, fat).from \land \\ \qquad\qquad fas2mms\, fat.to = (fat2mmt\, fat).to))) \end{array}$$

Corresponding transitions in the failure automaton and Markov model must map from and to corresponding states.

$$\begin{array}{|l} GetStructuralCorrespondence : FailureAutomaton \times MarkovModel \nrightarrow \\ \quad StateCorrespondence \times TransitionCorrespondence \\ \hline \mathrm{dom}\, GetStructuralCorrespondence = \\ \quad \{fa : FailureAutomaton;\ mm : MarkovModel \mid \\ \qquad (\exists fas2mms : StateCorrespondence;\ fat2mmt : TransitionCorrespondence \bullet \\ \qquad StructuralCorrespondenceExists(fa, mm)) \bullet (fa, mm)\} \\ \forall fa : FailureAutomaton;\ mm : MarkovModel; \\ \quad fas2mms : StateCorrespondence;\ fat2mmt : TransitionCorrespondence \mid \\ \quad (fa, mm) \in \mathrm{dom}\, GetStructuralCorrespondence \land \\ \quad \mathrm{dom}\, fas2mms = fa.states \land \mathrm{ran}\, fas2mms = mm.states \land \\ \quad \mathrm{dom}\, fat2mmt = fa.transitions \land \mathrm{ran}\, fat2mmt = mm.transitions \land \\ \quad (\forall fat : FailureAutomatonTransition \mid fat \in fa.transitions \bullet \\ \qquad (fas2mms\, fat.from = (fat2mmt\, fat).from \land \\ \qquad fas2mms\, fat.to = (fat2mmt\, fat).to)) \bullet \\ \quad GetStructuralCorrespondence(fa, mm) = (fas2mms, fat2mmt) \end{array}$$

This helper function computes the correspondence between a failure automaton and its associated Markov model.

10.2 Markov Model Transition Rate Functions

In this section we specify the transition rate functions for the Markov model transitions in terms of the transitions in the failure automaton. The basis for this transition rate function is the "hazard function" of the distribution associated with the causal basic event.

$$
\begin{array}{|l}
GetNonDeterministicTransitions : FailureAutomatonTransition \times \\
\quad FailureAutomaton \pfun \mathbb{F}\, FailureAutomatonTransition \\
\hline
\mathrm{dom}\, GetNonDeterministicTransitions = \\
\quad \{ fat : FailureAutomatonTransition; fa : FailureAutomaton \mid \\
\quad\quad fat \in fa.transitions \bullet (fat, fa) \} \\
\forall fat : FailureAutomatonTransition; fa : FailureAutomaton \mid \\
\quad (fat, fa) \in \mathrm{dom}\, GetNonDeterministicTransitions \bullet \\
\quad GetNonDeterministicTransitions(fat, fa) = \\
\quad\quad \{ t : FailureAutomatonTransition \mid t \in fa.transitions \land \\
\quad\quad t.causalBasicEvent = fat.causalBasicEvent \bullet t \}
\end{array}
$$

GetCausalEventsBetweenStates computes the set of causal events for the transitions between two states in the fa.

$$
\begin{array}{|l}
SparingScaleFactor : FailureAutomatonTransition \times FailureAutomaton \pfun \mathbb{R} \\
\hline
\mathrm{dom}\, SparingScaleFactor = \\
\quad \{ fat : FailureAutomatonTransition; fa : FailureAutomaton \mid \\
\quad\quad fat \in fa.transitions \bullet (fat, fa) \} \\
\forall fat : FailureAutomatonTransition; fa : FailureAutomaton \mid \\
\quad (fat, fa) \in \mathrm{dom}\, SparingScaleFactor \bullet \\
\quad SparingScaleFactor(fat, fa) = \\
\quad\quad 1_{\mathbb{R}} /_{\mathbb{R}} intToReal(\#(GetNonDeterministicTransitions(fat, fa)))
\end{array}
$$

The sparing scale factor is equal to the inverse of the number of nondeterministic next states that have the same set of causal basic events as between the from and to states.

$$
\begin{array}{|l}
CoverageScaleFactor : FailureAutomatonTransition \times \\
\quad BasicEventModelFunction \pfun \mathbb{R} \\
\hline
\mathrm{dom}\, CoverageScaleFactor = \\
\quad \{ fat : FailureAutomatonTransition; bemf : BasicEventModelFunction \mid \\
\quad\quad fat.causalBasicEvent \in \mathrm{dom}\, bemf \bullet (fat, bemf) \} \\
\forall fat : FailureAutomatonTransition; bemf : BasicEventModelFunction \mid \\
\quad (fat, bemf) \in \mathrm{dom}\, CoverageScaleFactor \bullet \\
\quad CoverageScaleFactor(fat, bemf) = \\
\quad\quad \textbf{if}\ fat.from.systemFailedUncovered = False \land \\
\quad\quad\quad fat.to.systemFailedUncovered = True \\
\quad\quad \textbf{then} \\
\quad\quad\quad (bemf\, fat.causalBasicEvent).coverageModel.singlePointFailureProbability \\
\quad\quad \textbf{else} \\
\quad\quad\quad (bemf\, fat.causalBasicEvent).coverageModel.coveredFailureProbability
\end{array}
$$

This function determines the coverage factor that should be applied to the transition rate function de-

pending on whether the state is failed uncovered.

$$NumberOfReplsInUse : Event \times FailureAutomatonState \rightarrow \mathbb{N}$$

$$\forall e : Event;\ fas : FailureAutomatonState \bullet$$
$$NumberOfReplsInUse(e, fas) = \#(fas.spareInUse \rhd \{e\})$$

The number of replicates of a basic event that are in use for a given FA state is the number of mappings from spare gates to spares such that the spares are restricted to be equal to the basic event.

$$NumberOfOperReplsNotInUse : Event \times FailureAutomatonState \times ReplicationMap \nrightarrow \mathbb{N}$$

$$\mathrm{dom}\, NumberOfOperReplsNotInUse =$$
$$\{ e : Event;\ fas : FailureAutomatonState;\ rs : ReplicationMap \mid$$
$$e \in \mathrm{dom}\, fas.stateOfEvents \bullet (e, fas, rs) \}$$
$$\forall e : Event;\ fas : FailureAutomatonState;\ rs : ReplicationMap \mid$$
$$(e, fas, rs) \in \mathrm{dom}\, NumberOfOperReplsNotInUse \bullet$$
$$NumberOfOperReplsNotInUse(e, fas, rs) =$$
$$rs\, e - fas.stateOfEvents\, e - NumberOfReplsInUse(e, fas)$$

The number of replicates of a basic event that are operational but not in use for a given FA state is the replication less the number of replicates that have already occurred less the number of replicates in use by spare gates.

$$DormancyReplicationScaleFactor :$$
$$FailureAutomatonTransition \times FailureAutomaton \times BasicEventModelFunction \times ReplicationMap \nrightarrow \mathbb{R}$$

$$\mathrm{dom}\, DormancyReplicationScaleFactor =$$
$$\{ fat : FailureAutomatonTransition;\ fa : FailureAutomaton;$$
$$bemf : BasicEventModelFunction;\ rs : ReplicationMap \mid$$
$$fat \in fa.transitions \land fat.causalBasicEvent \in \mathrm{dom}\, bemf \bullet$$
$$(fat, fa, bemf, rs) \}$$
$$\forall fat : FailureAutomatonTransition;\ fa : FailureAutomaton;$$
$$bemf : BasicEventModelFunction;\ rs : ReplicationMap \mid$$
$$(fat, fa, bemf, rs) \in \mathrm{dom}\, DormancyReplicationScaleFactor \bullet$$
$$DormancyReplicationScaleFactor(fat, fa, bemf, rs) =$$
$$intToReal(NumberOfReplsInUse(fat.causalBasicEvent, fat.from)) +_{\mathbb{R}}$$
$$intToReal(NumberOfOperReplsNotInUse(fat.causalBasicEvent,$$
$$fat.from, rs)) *_{\mathbb{R}} (bemf\, fat.causalBasicEvent).dormancy$$

The dormancy/replication scale factor is the number of replicates of the causal basic event that are in use by spare gates, plus the number of operational replicates that are not in use multiplied by the dormancy.

$$HazardFunction == Time \rightarrow Rate$$

A hazard function is a function of time, and is based on the distribution. We do not specify the details of the hazard function.

$$
\begin{array}{|l}
ComputeHazardFunction : Distribution \pfun HazardFunction \\
\hline
\dom ComputeHazardFunction = \\
\quad \{ d : Distribution \mid d \in exponentialDistributions \lor \\
\quad d \in weibullDistributions \bullet d \} \\
\forall d : Distribution; \; hf : HazardFunction \mid \\
\quad d \in \dom ComputeHazardFunction \bullet \\
\qquad hf = ComputeHazardFunction\, d
\end{array}
$$

ComputeHazardFunction computes the hazard function associated with a distribution. We abstract the details of this computation in this specification. However, an important constraint is that the hazard function can only be computed for exponential and Weibull distributions. This effectively constraints the fault trees whose corresponding failure automata semantics can be expressed in terms Markov models to the set of exponential or Weibull fault trees.

$$
\begin{array}{|l}
ScaledTransitionRateFunction : FailureAutomatonTransition \times \\
\quad FailureAutomaton \times BasicEventModelFunction \times \\
\quad ReplicationMap \pfun TransitionRateFunction \\
\hline
\dom ScaledTransitionRateFunction = \\
\quad \{ fat : FailureAutomatonTransition; \; fa : FailureAutomaton; \\
\quad bemf : BasicEventModelFunction; \; rs : ReplicationMap \mid \\
\quad fat \in fa.transitions \land fat.causalBasicEvent \in \dom bemf \bullet \\
\quad (fat, fa, bemf, rs) \} \\
\forall fat : FailureAutomatonTransition; \; fa : FailureAutomaton; \\
\quad bemf : BasicEventModelFunction; \; d : Distribution; \; rs : ReplicationMap \mid \\
\quad (fat, fa, bemf, rs) \in \dom ScaledTransitionRateFunction \land \\
\quad d = (bemf\, fat.causalBasicEvent).distribution \land \\
\qquad d \in \dom ComputeHazardFunction \bullet \\
\qquad ScaledTransitionRateFunction(fat, fa, bemf, rs) = \\
\qquad\quad ComputeHazardFunction\, d *_{pf(\mathbb{R})} CoverageScaleFactor(fat, bemf) *_{\mathbb{R}} \\
\qquad\quad DormancyReplicationScaleFactor(fat, fa, bemf, rs) *_{\mathbb{R}} \\
\qquad\quad SparingScaleFactor(fat, fa)
\end{array}
$$

A scaled transition rate function is the hazard function scaled by the various scale factors.

$$
\begin{array}{l}
SumScaledTransitionRateFunctions : \\
\quad \mathbb{F}\, FailureAutomatonTransition \times FailureAutomaton \times \\
\quad BasicEventModelFunction \times ReplicationMap \rightarrow TransitionRateFunction \\
\hline
\mathrm{dom}\, SumScaledTransitionRateFunctions = \\
\quad \{ fats : \mathbb{F}\, FailureAutomatonTransition;\ fa : FailureAutomaton; \\
\quad bemf : BasicEventModelFunction;\ rs : ReplicationMap \mid \\
\quad fats \subseteq fa.transitions \land \\
\quad (\forall fat : FailureAutomatonTransition \mid fat \in fats \bullet \\
\qquad (fat.causalBasicEvent \in \mathrm{dom}\, bemf)) \bullet \\
\quad (fats, fa, bemf, rs) \} \\
\forall fat : FailureAutomatonTransition;\ fa : FailureAutomaton; \\
\quad fats : \mathbb{F}\, FailureAutomatonTransition;\ bemf : Event \nrightarrow BasicEventModel; \\
\quad rs : ReplicationMap \mid (fats \cup \{fat\}, fa, bemf, rs) \in \mathrm{dom} \\
\qquad SumScaledTransitionRateFunctions \bullet \\
\qquad SumScaledTransitionRateFunctions(\{fat\}, fa, bemf, rs) = \\
\qquad\quad ScaledTransitionRateFunction(fat, fa, bemf, rs) \land \\
\qquad SumScaledTransitionRateFunctions(\{fat\} \cup fats, fa, bemf, rs) = \\
\qquad\quad ScaledTransitionRateFunction(fat, fa, bemf, rs) +_{pf(\mathbb{R})} \\
\qquad\quad SumScaledTransitionRateFunctions(fats, fa, bemf, rs)
\end{array}
$$

The sum of the scaled transition rate functions is computed from a set of failure automaton transitions recursively.

$$
\begin{array}{l}
ComputeTransitionRateFunction : \\
\quad FailureAutomatonTransition \times FailureAutomaton \times \\
\qquad BasicEventModelFunction \times ReplicationMap \nrightarrow TransitionRateFunction \\
\hline
\mathrm{dom}\, ComputeTransitionRateFunction = \\
\quad \{ fat : FailureAutomatonTransition;\ fa : FailureAutomaton; \\
\quad bemf : BasicEventModelFunction;\ rs : ReplicationMap \mid \\
\quad fat \in fa.transitions \land fat.causalBasicEvent \in \mathrm{dom}\, bemf \land \\
\quad (\forall fat_2 : FailureAutomatonTransition \mid \\
\qquad fat_2.from = fat.from \land fat_2.to = fat.to \bullet \\
\qquad (fat_2.causalBasicEvent \in \mathrm{dom}\, bemf)) \bullet \\
\quad (fat, fa, bemf, rs) \} \\
\forall fat : FailureAutomatonTransition;\ fa : FailureAutomaton; \\
\quad fats : \mathbb{F}\, FailureAutomatonTransition; \\
\qquad bemf : BasicEventModelFunction;\ rs : ReplicationMap \mid \\
\quad (fat, fa, bemf, rs) \in \mathrm{dom}\, ComputeTransitionRateFunction \land \\
\quad fats = \{ t : fa.transitions \mid t.from = fat.from \land t.to = fat.to \bullet t \} \bullet \\
\quad ComputeTransitionRateFunction(fat, fa, bemf, rs) = \\
\qquad SumScaledTransitionRateFunctions(fats, fa, bemf, rs)
\end{array}
$$

The transition rate function for a Markov model transition is the sum of the scaled transition rate functions.

10.3 Complete Failure Automaton Semantics in terms of Markov Models

$$
\begin{array}{|l}
\mathit{FailureAutomatonSemantics} : \\
\quad \mathit{FailureAutomaton} \times \mathit{BasicEventModelFunction} \times \\
\quad \mathit{ReplicationMap} \nrightarrow \mathit{MarkovModel} \\
\hline
\mathrm{dom}\, \mathit{FailureAutomatonSemantics} = \\
\quad \{ fa : \mathit{FailureAutomaton};\ bemf : \mathit{BasicEventModelFunction};\ rs : \mathit{ReplicationMap} \mid \\
\qquad \{ fat : \mathit{FailureAutomatonTransition} \mid fat \in fa.\mathit{transitions} \bullet \\
\qquad fat.\mathit{causalBasicEvent} \} \subseteq \mathrm{dom}\, bemf \bullet (fa, bemf, rs) \} \\
\forall fa : \mathit{FailureAutomaton};\ bemf : \mathit{BasicEventModelFunction}; \\
\quad mm : \mathit{MarkovModel};\ rs : \mathit{ReplicationMap} \mid \\
\qquad (fa, bemf, rs) \in \mathrm{dom}\, \mathit{FailureAutomatonSemantics} \bullet \\
\quad \mathit{FailureAutomatonSemantics}(fa, bemf, rs) = mm \Leftrightarrow \\
\qquad \mathit{StructuralCorrespondenceExists}(fa, mm) \wedge \\
\qquad (\exists fas2mms : \mathit{StateCorrespondence};\ fat2mmt : \mathit{TransitionCorrespondence} \mid \\
\qquad (fas2mms, fat2mmt) = \mathit{GetStructuralCorrespondence}(fa, mm) \bullet \\
\qquad\quad (\forall fat : \mathit{FailureAutomatonTransition} \mid fat \in fa.\mathit{transitions} \bullet \\
\qquad\quad (fat2mmt\, fat).\mathit{transitionRateFunction} = \\
\qquad\qquad \mathit{ComputeTransitionRateFunction}(fat, fa, bemf, rs)))
\end{array}
$$

The complete failure automaton semantics is specified as a correspondence between states and transitions, and the computation of the transition rate function from the basic event model functions, the replications, and the transitions in the models.

11 Analyses

A common analysis for fault trees is the determination of overall system unreliability. In this section we present a formal specification of that analysis.

$$
\begin{array}{|l}
IsSystemFailedState\_ : \mathbb{P}(MarkovModelState \times Event \times \\
\quad FailureAutomaton \times StateCorrespondence) \\
\hline
\forall ms : MarkovModelState;\ se : Event;\ fa : FailureAutomaton; \\
\quad fas2mms : StateCorrespondence;\ fas : FailureAutomatonState \mid \\
\quad se \in \mathrm{dom} fas.stateOfEvents \land ms = fas2mms\, fas \bullet \\
\quad IsSystemFailedState(ms, se, fa, fas2mms) \Leftrightarrow \\
\quad\quad fas.stateOfEvents\, se > 0 \lor fas.systemFailedUncovered = True
\end{array}
$$

This helper function determines whether a particular state in the Markov model is a system failed state.

$$
\begin{array}{|l}
SystemUnreliability : FaultTree \times Event \times \\
\quad BasicEventModelFunction \times Time \rightarrow Probability \\
\hline
\forall ft : FaultTree;\ se : Event;\ bemf : BasicEventModelFunction;\ t : Time \bullet \\
\quad \exists fa : FailureAutomaton;\ mm : MarkovModel \mid \\
\quad fa = FaultTreeSemantics(ft) \land \\
\quad\quad mm = FailureAutomatonSemantics(fa, bemf, ft.replications) \land \\
\quad \{fat : FailureAutomatonTransition \mid \\
\quad\quad fat \in fa.transitions \bullet fat.causalBasicEvent\} \subseteq \mathrm{dom}\, bemf \bullet \\
\quad \exists fas2mms : StateCorrespondence;\ fat2mmt : TransitionCorrespondence \mid \\
\quad (fas2mms, fat2mmt) = GetStructuralCorrespondence(fa, mm) \bullet \\
\quad SystemUnreliability(ft, se, bemf, t) = \\
\quad\quad +/_{\mathbb{R}} \{ms : MarkovModelState \mid \\
\quad\quad\quad ms \in mm.states \land IsSystemFailedState(ms, se, fa, fas2mms) \bullet \\
\quad\quad\quad ms.finalProbability\}
\end{array}
$$

The system unreliability can be computed from a given fault tree *ft*, an identified "system level event", the basic event models, and a mission time. The overall system unreliability is the sum of the final probabilities of all Markov states that correspond to fault tree states that are failed. The failure automaton semantics will only be valid if the basic event models have distributions that have the Markov property.

Acknowledgements

This work was supported in part by the National Science Foundation under grant CCR-9804078 (Kevin Sullivan, principal investigator).

A Fault Tree Subtypes

Each of these sub-classes of fault trees correspond to particular sub-classes of failure automata via the *FaultTreeToFailureAutomaton* relation. We now formalize these sub-classes of fault trees.

StaticFaultTree

$$FaultTree$$

$$pandGates = \varnothing \land fdeps = \varnothing \land spareGates = \varnothing \land seqs = \varnothing$$

Static fault trees have no order-dependent constructs.

$$DynamicFaultTree \mathrel{\widehat{=}} \neg StaticFaultTree$$

Dynamic fault trees have one or more dynamic constructs.

B Bibliography

[1] J. M. Spivey. *The Z Notation: A Reference Manual.* Prentice Hall International Series in Computer Science, 2nd edition, 1992.