

Formal Specification in Collaborative Design of Critical Software Tools

David Coppit and Kevin J. Sullivan
March 11, 2005

Problem

- Models support design of critical systems
- Modeling tools are critical components
- These tools are complex software systems
- Evidence indicates critical tools have significant errors [Hatton, NRC, Murphy]

Assumptions & Approach

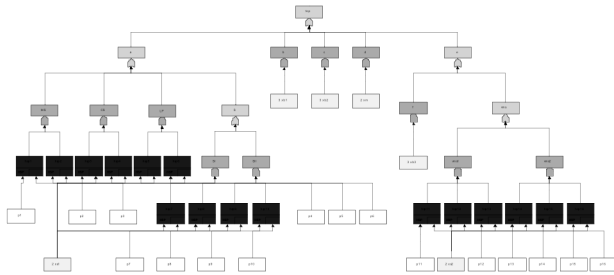
- Domain experts know modeling techniques
- Software engineers understand software
- Need to use rigorous software engineering in the collaborative design of critical tools
- We present a small case study

Case Study

- Dynamic Fault Tree Analysis [Dugan]
- Novel & complex modeling constructs are essential to both methodology and tool
- Why trust complex modeling framework?
- Why trust a given model of a given system?
- Why trust complex tool implementation?
- Collaborative development of partial formal specification of modeling framework

Example Fault Tree – Asid-Mas

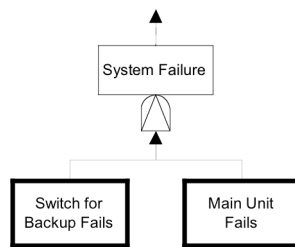
Advanced System Integration Demonstration –
Mission Avionics Subsystem



Modeling Constructs

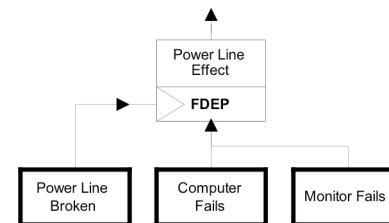
- Basic Events (distributions, replication)
- Static Gates: AND, OR, KofM
- Dynamic Gates: PAND, FDEP, SEQ, CSP, WSP, HSP

Priority AND Gate (PAND)



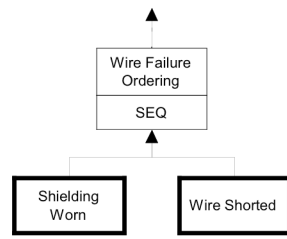
Fails if all the inputs fail in order

Functional Dependency (FDEP)



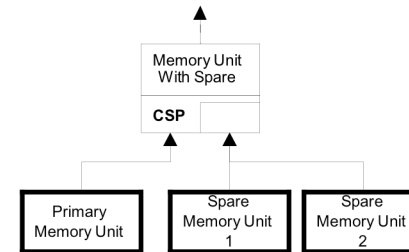
- Failure of trigger causes dependent inputs to fail
- Dummy output

Sequence Enforcing (SEQ)



- Expresses infeasible orderings
- Dummy output

Spare Gates (CSP, WSP, HSP)

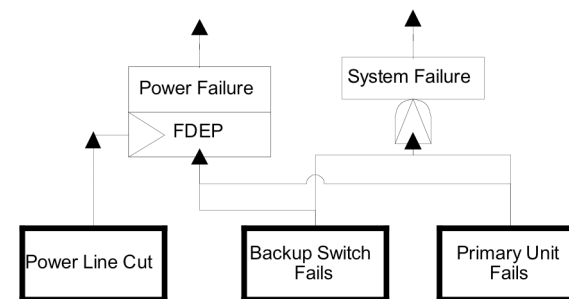


- If primary fails, spares used until none left, then gate fails
- Spares may be shared among spare gates

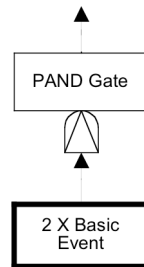
Subtleties in Dynamic Trees

- Timing of failures due to FDEP
- Semantics of failure of PAND
- Semantics of the SEQ
- Semantics of shared, pooled spares
- Failure status of replicated basic events

FDEP and PAND



PAND and Replicated Basic Event



Why Mathematical Specification?

- **Forces careful thought and documentation**
- Ease validation of modeling framework
- Ease validation of models against systems
- Ease verification of implementation
- Amenable to syntax/semantics checking

But is collaboration on specifications profitable?

Outline of the Specification

- Probability distributions
- Basic events
- Gates
- Valid fault tree

Introduction to Z

- Based on sets ($\cap \cup \in \notin$), logic ($\neg \wedge \vee \Rightarrow \forall \exists$)
- Typed sets are the basic definition mechanism
- Every value has exactly one type
- Structuring mechanism: the *schema calculus*

Spider	
legs	: \mathbb{N}
eyes	: \mathbb{N}
legs = 8	

Specification of PAND

```

PAND
Gate
inputOrder : iseq FaultTreeNodes

output ≠ dummy
ran inputOrder = inputs
∀inputNode: inputs | inputNode ∈ BasicEvents •
  (GetBasicEvent(inputNode)).replication = 1
  ...
∧ ((∃inputNodeA, inputNodeB: inputs |
  inputOrder~(inputNodeA) < inputOrder~(inputNodeB) •
  (inputNodeA ∈ Gates ∧ GetOutput(inputNodeA) = failed
  ∧ inputNodeB ∈ Gates ∧ GetOutput(inputNodeB) = failed
  ∧ (GetGate(inputNodeA)).failTime ≥ (GetGate(inputNodeB)).failTime))
⇒ output = operational)
  ...

```

Specification Problems Found

- PAND gate can not have a replicated basic event as input
- A basic event is not operational or failed because of replication; it depends on the gate that uses it
- Simultaneous failure of SEQ gate's inputs should be valid
- Sharing of spares among spare gates of different types should be invalid

Implementation Problems Found

- Cascaded FDEPs not handled correctly
- PAND gate's output is not consistently operational for simultaneous failure of inputs
- If more than one spare gate needs a shared spare, an arbitrary decision is made as to who gets it
- Sharing of primaries is allowed

Conclusion

- Engineering tools used to model and analyze critical systems
- Tools are complex software systems subject to specification and implementation errors
- Modern formal specification techniques can help avert some errors
- Collaboration on formal specifications with domain experts can be possible & profitable