Automatic Generation of Goal-Tree from Statecharts
Requirements Specification

June 3, 2003

Joonbeom Yoo*, Sungdeok Cha*, and Han Seong Son**
*KAIST, **KAERI
Daejeon, Korea
Towards Automatic Generation of Fault Trees from Statecharts Requirements Specification

- Statecharts
- An Example: KNICS DPPS Trip Logic
- Related Work
- Proposed Approach
- Conclusion and Future Work

Work in progress...
Statecharts

- Formal and visual requirements specification language
  - Activity Chart + Statechart + Module Chart

- Widely used for reactive systems
  - Hierarchical Structure
  - Concurrency
  - Broadcasting

**Statemate:**
- Graphical editors, data dictionary, simulator, model checker, ...
Statemate and Activity Chart
In KNICS project, Statecharts are being used to specify SRS for ESF-CCS, PLC application software, ...

KINS, regulatory body, requires safety analysis (FTA) on SRS.
Fixed Set-Point Rising Trip logic (simplified)
- Trip_By_Logic: If trip condition is continued for predefined time
- Trip_By_Error: If external input data has invalid value

```
Get_Input
Valid

\[ f_X \geq \text{Min and } f_X \leq \text{Max} \]

\[ f_X < \text{Min or } f_X > \text{Max} \]

Waiting
Normal

\[ \text{tm}(\text{en(Waiting)}, \text{DelayTime}) \]

\[ f_X < \text{Setpoint - Hys} \]

\[ f_X \geq \text{Setpoint} \]

\[ f_X < \text{Setpoint} \]

\[ f_X = \text{Setpoint} \]

Invalid
Valid
Normal

\[ \text{Trip_Logic} = 0 \]

\[ \text{Trip_Logic} = 1 \]

\[ \text{Trip_Logic} = 0 \]

\[ \text{Trip_Logic} = 1 \]
```

Diagram:
- Trip_Firing
  - Normal (3.1)
    - Valid (1.2)
    - Invalid / Trip (3.2)
    - Trip_By_Error (3.3)
    - Trip_By_Logic (3.3)
  - Trip_By_Error (3.2)
    - Valid / Normal
- Input_By_Error
  - Get_Input (1.1)
    - Valid (1.2)
    - Invalid (1.3)
    - C
    - Input
    - / Valid
    - / Invalid
- Trip_By_logic
  - Normal (2.1)
  - Waiting (2.2)
  - Trip_By_Error (3.2)
    - Input
    - \[ f_X = \text{Setpoint} \]
    - tm(en(Waiting), DelayTime)
    - / Trip_Logic = 1
  - Trip_By_Logic (3.3)
    - Input
    - \[ f_X < \text{Setpoint - Hys} \]
    - / Trip_Logic = 0
```
Related Work: [Mojdehbakhsh, et al. (1994)]

- Primarily concerned with how overall software requirements safety analysis process is to be applied
Related Work: [Mojdehbakhsh, et al. (1994)]

5. For each entity identified in 3.2 do the following:
5.1. Create an AND node labeled event.x where x = 1..n. Connect this node as a child to the node created in 1.
5.2. Set the right child of this node to in(state-name)
5.3. Create an OR node for the left child labeled event.trigger
5.4. Identify all actions in this entity that generate the event.
5.5. Identify all independent triggers of these actions.
5.6. Assign the items in 5.5 as the children of the node in created 5.3.

- No concrete guidelines are given as to how steps 5.4 and 5.5 could be applied systematically
Related Work: [Mojdehbakhsh, et al. (1994)]

Figure 5. Template for single primitive event. The three dots represent repetition of nodes. The left branch of the tree is a template for entities in rule 3.2, and the middle branch for those in 3.1. There can be as many event.n nodes as required for both 3.1 and 3.2 type entities.
Related Work

FT Generated by [Mojdehbakhsh, et al. (1994)]

No guidelines as to how to further analyze credible causes for the event “Invalid”

State-oriented analysis

```
Transition from 3.2 to 3.1 enabled
```

```
Transition from 3.3 to 3.1 enabled
```

```
Timeout occurs
```

```
Input \{f_X \geq \text{Setpoint}\}
```

```c
in Normal (3.1)
```

```c
in Trip_By_Error (3.2)
```

```c
in Trip_By_Logic (3.3)
```

```
in Normal (3.1)
```

```c
Invalid
```

```c
Valid
```

```c
in Waiting (2.2)
```

```c
in Normal (2.1)
```

```c
in Normal (2.1)
```

```c
Timeout occurs
```

```c
Input \{f_X \geq \text{Setpoint}\}
```
Related Work: [Ratan, et al. (1996)]

Figure 5. Basic template for each level of fault tree.

Figure 6. Expansion of a previous configuration node.
Related Work: [Ratan, et al. (1996)]

RSML vs Statecharts

Figure 3. An AND/OR table.
Related Work: [Ratan, et al. (1996)]

**Figure 7.** Expansion of guarding conditions information node.
Related Work

FT Generated by [Ratan, et al. (1996)]

Further analysis for event “Invalid” IMO, poor readability.
Proposed Approach

Step 1
Pre-processing (done manually)

Identify external activity(s)
Select the top-level event

External activity(s): event Input, data \( f_X \)
Top-hazard event: Trip due to Invalid Event
Proposed Approach

Step #2: Transform Statecharts to ...

* Dependency analysis, program slicing
Proposed Approach

Step #2: Statecharts -> Labeled Digraph

Invalid

Was in Invalid state
(1.3)

[f_X < Min or f_X > Max]

External event occurs

Input

[f_X < Min]

[f_X > Max]

[f_X < Min]

[f_X > Max]

In Get_Input state
(1.1)
Systematic Event-based Fault-Tree Generation

Step 2: Statecharts → Labeled Digraph

- Start
- Assign the top-hazard action to the terminal node
- There are both event and condition that trigger the previous node
- Make an AND gate
- Find the source node occurring the event (or condition). And assign the node to the one-step-back node.

- Is it a complex node?
  - Yes
    - Is it CNF?
      - Yes
        - Make an AND gate
      - No
        - Assign the single node(s) to the one-step-back node.
  - No
    - Completed
Systematic Event-based Fault-Tree Generation

Step 2: Statecharts → Labeled Digraph
Proposed Approach

Step 3: Labeled Digraph to Fault Tree

---

Step 3
Labeled Digraph → Fault-Tree

---

[Diagram of labeled digraph and its transformation into a fault tree]

Invalid

Get_Input (1.1)

Input

[ f_X < Min or f_X > Max ]

[ f_X < Min ]

[ f_X > Max ]

[ f_X < Min ]

[ f_X > Max ]

[ f_X < Min or f_X > Max ]

---

20
Systematic Event-based Fault-Tree Generation

Step 3: Labeled Digraph → Fault-Tree Patterns

1. Top-Hazard event node (Terminal node)

2. AND-gate

3. External activity node (Single event)

4. External activity node (Single condition)

5. State (Single event: time-out)

6. State (Single condition: in(...))

7. Complex condition (logical AND)

8. Complex condition (logical OR)

9. Variable (a single variable of complex condition)

10. Complex event (logical OR)
Systematic Event-based Fault-Tree Generation

Step 4: Fault Tree Reduction, Cycle Resolution, ...

“Macro definition” of common fault trees,

Resolution of cyclic fault trees
(primarily because Statecharts are most likely to exhibit cyclic system behavior),

Minimal cut-set analysis,

...
Proposed Approach

FT Generated by Proposed Approach

Further analysis for event "Invalid"
Summary

We’re trying to propose a systematic process for developing fault-tree from Statecharts.

- “More than ‘too-elementary-to-be-useful’ guidelines”
- “Improved understandability”
- Backward analysis becomes quite complex when “domino effects” and complex guarding conditions are considered

Accuracy of fault tree can be easily (albeit manually) verified by human experts
Future Work

Process is not yet fully refined/automated

Critical (and empirical) comparison of competing approaches by domain experts