The Mechanical Generation of Fault Trees for State Transition Systems

Richard Banach
School of Computer Science, University of Manchester, UK

Marco Bozzano
Fondazione Bruno Kessler, FBK-IRST, Trento, Italy
Contents:

1. Overview of FTA.
2. Model-Based Safety Analysis.
3. The FSAP Safety Analysis Platform.
5. FT Extraction for Combinational Circuits.
7. Clocked Acyclic Circuits, Causal Relations, Retrenchments.
8. FT Extraction for Clocked Acyclic Circuits.
10. FT Extraction for Feedback Circuits.
11. FSAP and the Model Checking Approach to FT Extraction.
12. Retrenchment and Model Checking Compared.
1. Overview of FTA.

Fault Tree Analysis (FTA) is a traditional safety analysis activity.

Main features:
• Deductive technique.
• Graphical representation of the effects of failures on system requirements. (Boolean gates to represent the logical interrelationships between events)
• Widespread use in aerospace, automotive, nuclear power plants, etc.
• Qualitative model that can be evaluated quantitatively.

In the rest of this chapter:
• Short introduction to safety analysis and FTA.
• Fault tree basics.
• Not an exhaustive presentation on FTA: mainly the notions needed in the rest of the tutorial will be presented.
Motivations

Objectives of safety analysis:
• Determine the conditions under which safety hazards can occur.
• Ensure that a system meets the safety requirements that are required for its deployment and use.

Particularly important for safety-critical systems, where unexpected behaviour may cause significant loss of money or human lives!

Safety levels can be domain-dependent: e.g., notion of fail-safe state in railways (all trains stopped, all signals at red), but no fail-safe state in avionics.
Motivations

Safety analysis:
- Typically needed for certification of safety-critical systems.

Safety analysis must:
- Analyse system behaviour under all possible operational conditions.
- In particular in presence of malfunctions of its components.
Safety Analysis

System Design
- System Level Requirements
- System Architecture
- System Implementation

Safety Analysis
- Fault Hazard Analysis
- PSSA
- System Safety Analysis

Certification

Complex System

R. Banach, School of Computer Science, University of Manchester, UK
M. Bozzano, Fondazione Bruno Kessler, FBK-IRST, Trento, Italy
Safety Analysis

System Design
- System Level Requirements
- System Architecture
- System Implementation

Safety Analysis
- Fault Hazard Analysis
- PSSA
- System Safety Analysis

Certification

Fault Tree Analysis (FTA)

Failure Mode and Effects Analysis (FMEA)

<table>
<thead>
<tr>
<th>Fault</th>
<th>Hazard</th>
<th>Consequence</th>
<th>Final Effect</th>
<th>Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undetected fire in bay area</td>
<td>Level 1</td>
<td>Subsystem A fails</td>
<td>Loss of mechanical drive</td>
<td>5</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

R. Banach, School of Computer Science, University of Manchester, UK
M. Bozzano, Fondazione Bruno Kessler, FBK-IRST, Trento, Italy
Safety Analysis

Safety Assessment carried out in parallel with system design and development.

E.g., safety assessment process model in avionics.

Several safety assessment activities, e.g.:
- Fault Hazard Analysis (FHA).
- Event Tree Analysis.
- Failure Mode and Effects Analysis (FMEA).
- Fault Tree Analysis (FTA).
- …

Fault trees produced at different stages of safety assessment.

R. Banach, School of Computer Science, University of Manchester, UK
M. Bozzano, Fondazione Bruno Kessler, FBK-IRST, Trento, Italy
Safety Analysis

An example – safety property (qualitative):
- “If no more than 3 components fail, then I never have a total loss of hydraulic power”.
- “No single point of failure can cause unavailability of both the primary and secondary power systems”.

An example – safety property (quantitative):
- “The probability of a total loss of hydraulic power is less than $10^{-7}$”.
- “The probability that both the primary and secondary power systems fail during the same mission is less than $10^{-9}$”.
Safety Analysis

An example – Fault Tree Analysis:

• “Find all combinations of basic faults which may cause total loss of hydraulic power”.

Particular interest in single points of failure … more in general in minimal combinations of faults.

Combination of basic faults = cut set.
Minimal combination = minimal cut set.
Fault Tree Analysis

Top Level Event (TLE)...

R. Banach, School of Computer Science, University of Manchester, UK
M. Bozzano, Fondazione Bruno Kessler, FBK-IRST, Trento, Italy
Fault Tree Analysis

Top Level Event (TLE) ...

... may be caused by:

R. Banach, School of Computer Science, University of Manchester, UK
M. Bozzano, Fondazione Bruno Kessler, FBK-IRST, Trento, Italy
Fault Tree Analysis

Top Level Event (TLE)

... may be caused by:

Minimal Cut Set 1
Fault Tree Analysis

Top Level Event (TLE)

... may be caused by:

Minimal Cut Set 1 or Minimal Cut Set 2
Fault Tree Analysis

Top Level Event (TLE)

... may be caused by:

Minimal Cut Set 1 or
Minimal Cut Set 2 or

...
Fault Tree Basics

A fault tree involves:

- Specifying a top level event (TLE) representing an undesired state.
- Find all possible chains of basic events that may cause the TLE to occur.

A fault tree:

- Is a systematic representation of such chains of events.
- Uses logical gates to represent the interrelationships between events and TLE, e.g. AND, OR.

An example fault tree
Logically: \((A \lor (B \lor C)) \land (C \lor (A \land B))\)
Fault Tree Basics

Logically, fault trees are equivalent if the associated logical formulae are equivalent.

E.g., \[(A \lor (B \lor C)) \land (C \lor (A \land B))\] \(\equiv\) \[(C \lor (A \land B))\]
Minimal Cut Sets

This shape is of particular interest – representation in terms of Minimal Cut Sets (MCS).

Minimal cut set = “smallest set of basic events which, in conjunction, cause the top level event to occur”.

Logically: Disjunctive Normal Form (DNF) = disjunction of conjunctions of basic events.

The fault tree on the left has two minimal cut sets: C (single point of failure) and A \land B (cut set of order 2).
Fault Tree Concepts

**Boundary** of the analysis: e.g., FTA performed at the system or sub-system level. **Resolution** of the analysis (abstraction and refinement techniques may be used).

It is up to the safety engineer to decide the set of basic events, depending on the boundary and the level of resolution of the analysis.

Rule of development: *identify the immediate, necessary and sufficient causes* for the occurrence of an event.
Fault Tree Concepts

A proper choice of intermediate events and the way the events are connected make the fault tree meaningful, not only the logical interrelationships.

No unique choice of intermediate events: e.g., they may be suggested by the structure of the system (“fault due to primary sub-system”, “fault due to secondary sub-system”) or the fault type (“system internal failure”, “system operated improperly”).

No unique way to build a fault tree …
Fault Tree Concepts

Fault trees are a qualitative model – but they can be evaluated quantitatively. Example of fault tree with attached probabilities:
Fault Tree Concepts

Questions that fault trees can answer – qualitative:
• Check if the top level event is reachable.
• Finding all the minimal cut sets causing the top level event.
• Check if there are single points of failure, i.e., minimal cut sets of order one.
• List all minimal cut sets of order one or two.
• ...

Questions that fault trees can answer – quantitative:
• Calculate the probability of top level event to occur.
• Check if there is any cut set with probability higher than $10^{-7}$.
• List all minimal cut sets with probability higher than $10^{-7}$.
• ...

In the rest of the tutorial: focus on fault trees as a qualitative model.
Fault Tree Concepts

Why fault trees are useful:
- They help understanding the system under analysis.
- They may reveal safety and reliability issues early in the design process.
- They may be used as a diagnostic tool, to identify and correct problems.
- They may assist engineers in design allocation.
- They may assist engineers in the evaluation of design alternatives or design upgrades.
- They may help in reducing design costs.
Fault Tree Extensions

Some topics that will not be discussed in this tutorial:

- A plethora of gates other than Boolean ones: inhibit, combination, priority AND, …
- Fault tree evaluation and reliability models: reliability function, probability density, failure rate.
- Dynamic fault trees: sequence dependencies, coverage modeling.
- In-depth discussion about causality.
2. Model-Based Safety Analysis.

Traditional analysis:
- Typically performed manually.
- Rely of the skills of safety engineers.
- Error-prone.

The model-based paradigm:
- Effort is re-directed to building models.
- Formal methods used to build both the system model and the fault model.
- Formal methods to elicit and write system requirements.
- Automated verification using formal methods techniques (e.g. model checking).
Model-Based Safety Analysis

Advantages:

- Sharing of information between design and safety assessment.
- Tighter integration of system design and safety analysis.
- Integration in the development cycle.
- Traceability & reusability.
- Unambiguous specification of the system and of the required properties.
- Exhaustive analysis.
- Automated generation of artifacts (e.g., fault trees).
- Improved effectiveness of the verification and validation process.
Model-Based Safety Analysis

Ideas pioneered by the ESACS and ISAAC projects.
  (EU-sponsored projects in FP5 and FP6)

Follow-up project MISSA.
  (EU-sponsored projects in FP7)

The ESACS and ISAAC Projects …

**ESACS (Enhanced Safety Assessment for Complex Systems)**

**IS AAC (Improvement of Safety Activities on Aeronautical Complex Systems)**

... and the MISSA Project

**MISSA (More Integrated Systems Safety Assessment)**

FP7 project - Duration: 04/2008 – 03/2011

R. Banach, School of Computer Science, University of Manchester, UK
M. Bozzano, Fondazione Bruno Kessler, FBK-IRST, Trento, Italy
The ESACS / ISAAC Methodology

Application Field:
Development process of Complex Systems used in safety critical industrial applications (in particular in the aeronautic field).

System Design
- System Level Requirements
- System Architecture
- System Implementation

Safety Analysis
- Functional Hazard Analysis
- Preliminary System Safety Assessment
- System Safety Assessment

Certification
Complex System

R. Banach, School of Computer Science, University of Manchester, UK
M. Bozzano, Fondazione Bruno Kessler, FBK-IRST, Trento, Italy
The ESACS / ISAAC Methodology

**Application Field:**
Development process of Complex Systems used in safety critical industrial applications (in particular in the aeronautic field).

**Goals:**
Improvement of the Safety Analysis practice on Complex Systems through the set-up of a shared environment between safety and design processes supported by tools based on Formal Methods and Verification Techniques.

R. Banach, School of Computer Science, University of Manchester, UK
M. Bozzano, Fondazione Bruno Kessler, FBK-IRST, Trento, Italy
The ESACS / ISAAC Methodology

Application Field:
Development process of Complex Systems used in safety critical industrial applications (in particular in the aeronautic field).

Goals:
Improvement of the Safety Analysis practice on Complex Systems through the set-up of a shared environment between safety and design processes supported by tools based on Formal Methods and Verification Techniques.

To reach the ESACS objective a new methodology has been defined (the ESACS methodology) and a platform (the ESACS platform) with tools supporting the methodology has been set-up.
Model-Based FTA

Model-based Fault Tree Analysis main concepts:

- Faults and fault models.
- Fault injection (automated model extension).
- Fault Tree generation based on fault injection.

In the following:

- Model-based safety analysis exemplified by the FSAP platform.
- FSAP is a safety analysis platform implementing the ESACS/ISAAC methodology.
- Demo of FSAP will follow.
Faults and Fault Models

Different fault models, depending on fault type and fault activation model.

Examples of fault types:
- “Stuck at”, “inverted”, “non deterministic”, “ramp down”, …
- Failure modes can be parametric, e.g. “stuck at value” failure).

Fault activation models:
- Permanent (once failed, always failed).
- Sporadic or transient (may present occasionally, or may be repaired).
Fault injection

Starting point: a System Model (SM) written in a formal language.
(Describes the nominal behaviour of the system)
• E.g. the NuSMV language in FSAP.

Definition of failure modes can be extracted for a failure model library.
• E.g. GFML (Generic Failure Mode Library) in FSAP.

Faults can be injected into the system model to allow for degraded behaviour.

Failure mode identification and characterization is tool independent.
Fault injection

Fault injection in FSAP:

System Model of Block “A”

Block “A” extended with two failure modes FM1 and FM2

R. Banach, School of Computer Science, University of Manchester, UK
M. Bozzano, Fondazione Bruno Kessler, FBK-IRST, Trento, Italy
Model Extension

Model extension is the process of injecting a set of component failure modes into the system model.

The result of the model extension is, again, a model written in a formal language. (Describes the possibly degraded behaviour of the system)
- E.g. the NuSMV language in FSAP.

The model with the injected faults is called Extended System Model (ESM).
Model Extension

Model extension in FSAP:

- Failure modes definition
- Generic Failure Mode Library
- ESM Generator
- System Model
- NuSMV SM
- NuSMV ESM
- Extended System Model

R. Banach, School of Computer Science, University of Manchester, UK
M. Bozzano, Fondazione Bruno Kessler, FBK-IRST, Trento, Italy
Fault Tree Generation

Model-based FTA: automated generation of fault trees based on fault injection. Inputs: an Extended System Model (ESM) and a top-level event (TLE). Outputs: fault trees and traces.

Fault tree generation in FSAP:
Fault Tree Generation

Fault tree generation in FSAP: traces are associated to minimal cut sets:

--- NuSMV Trace File --- ***** ---
################ Trace number1: **********

-> State 1.1 <-
VAR_1 = value_11
................
VAR_n = value_1n

-> State 1.2 <-
................

-> State 1.k <-
VAR_1 = value_k1
................
VAR_n = value_kn

FM_VAR_NAME_1 = VALUE_1
FM_VAR_NAME_2 = VALUE_2
FM_VAR_NAME_3 = VALUE_3
Fault Tree Generation

Further extensions available in FSAP:

- Definition of failure sets: group of failures that are activated simultaneously or in a user-specified order. (useful to model and analyse common-cause effects)

- Fault tree evaluation, based on a simple model of probability. (Hypothesis: independence of failures – except for common causes)

- Ordering analysis: analyse order between basic events in a cut set.
3. The FSAP Safety Analysis Platform.

R. Banach, School of Computer Science, University of Manchester, UK
M. Bozzano, Fondazione Bruno Kessler, FBK-IRST, Trento, Italy
The FSAP Safety Analysis Platform

Safety Analysis Platform:
- Developed at FBK.
- Under active development.

Composed of:
- FSAP (Graphical front-end).
- NuSMV-SA (Symbolic Model Checker).

Cross platform (Windows and Linux). Implemented in C++, with the FLTK graphical toolkit and the EXPAT library for XML parsing.
The FSAP Safety Analysis Platform

Provides:

- Simulation.
- Property verification.
- FTA.
- FMEA.
- Ordering analysis.
- FDIR.
- BDD- and SAT-based algorithms.

Furthermore:

- Generic Failure Mode Library.
- Data dictionary.
- Pattern-based safety requirements.

http://sra.itc.it/tools/FSAP/
The FSAP Safety Analysis Platform

Based on the NuSMV model checker:

- A powerful model checking tool.
- Integrates different engines: BDD-based, SAT-based.
- Robust, open, customizable.
- Developed under an OpenSource model, distributed under LGPL.
- Widely distributed and used: more than 500 installations worldwide.
- Used for teaching and in several industrial technology transfer projects.
- Interest expressed by various industrial partners and academics.

R. Banach, School of Computer Science, University of Manchester, UK
M. Bozzano, Fondazione Bruno Kessler, FBK-IRST, Trento, Italy
The FSAP Methodology

1. Model Definition
2. FM capturing
3. Model Extension
4. SR capturing
5. Model Analysis and Verification
6. Results Presentation
The FSAP Methodology

1: Model written in a formal language

FM capturing → Model Definition → Model Extension → Model Analysis and Verification → Results Presentation

R. Banach, School of Computer Science, University of Manchester, UK
M. Bozzano, Fondazione Bruno Kessler, FBK-IRST, Trento, Italy
The FSAP Methodology

2: Definition of failure modes, taken from a library

- FM capturing
- SR capturing

Model Definition
Model Extension
Model Analysis and Verification
Results Presentation

R. Banach, School of Computer Science, University of Manchester, UK
M. Bozzano, Fondazione Bruno Kessler, FBK-IRST, Trento, Italy
The FSAP Methodology

3: Automatic model extension: model + failure modes

FM capturing → 3 → Model Extension → Model Analysis and Verification → Results Presentation

SR capturing

R. Banach, School of Computer Science, University of Manchester, UK
M. Bozzano, Fondazione Bruno Kessler, FBK-IRST, Trento, Italy
The FSAP Methodology

4: Definition of safety requirements

- FM capturing
  - Model Definition
    - Model Extension
      - Model Analysis and Verification
        - Results Presentation

R. Banach, School of Computer Science, University of Manchester, UK
M. Bozzano, Fondazione Bruno Kessler, FBK-IRST, Trento, Italy
The FSAP Methodology

5: Model verification, FTA, FMEA, …

FM capturing → Model Definition

SR capturing → Model Extension

Model Analysis and Verification

Results Presentation

R. Banach, School of Computer Science, University of Manchester, UK
M. Bozzano, Fondazione Bruno Kessler, FBK-IRST, Trento, Italy
The FSAP Methodology

6: Display of results

FM capturing -> Model Definition

Model Extension

SR capturing -> Model Analysis and Verification

Results Presentation
The FSAP Architecture

R. Banach, School of Computer Science, University of Manchester, UK
M. Bozzano, Fondazione Bruno Kessler, FBK-IRST, Trento, Italy
The FSAP Architecture

Model Capturing
- Text Editor

FM Capturing
- FM Editor
- GFML

SR Capturing
- SR Editor
- GSRL

Safety Analysis Tools
- FT Displayer
- Sim Displayer
- FT Plus

Safety Result Extraction

SAT-Repository
- SAT Management
- SAT-DB

Model Analysis
- ESM Generator
- NuSMV-SA

R. Banach, School of Computer Science, University of Manchester, UK
M. Bozzano, Fondazione Bruno Kessler, FBK-IRST, Trento, Italy
The FSAP Architecture

Model Capturing
- Text Editor

FM Capturing
- FM Editor
- GFML

SR Capturing
- SR Editor
- GSRL

SAT-Repository
- SAT Management
- SAT-DB

Safety Analysis Tools
- FT Display
- FT Plus
- Sim Display

Safety Result Extraction
- NuSMV-SA
- ESM Generator

R. Banach, School of Computer Science, University of Manchester, UK
M. Bozzano, Fondazione Bruno Kessler, FBK-IRST, Trento, Italy
The FSAP Architecture

Model Capturing
- Text Editor

Safety Analysis Tools
- FT Display
- Sim Display
- FT Plus

FM Capturing
- FM Editor
- GFML

SR Capturing
- SR Editor
- GSRL

SAT-Repository
- SAT Management
- SAT-DB

Model Analysis
- ESM Generator

Safety Result Extraction
- NuSMV-SA

R. Banach, School of Computer Science, University of Manchester, UK
M. Bozzano, Fondazione Bruno Kessler, FBK-IRST, Trento, Italy
The FSAP Architecture

Model Capturing
- Text Editor

FMCapturing
- FM Editor
- GFML

SR Capturing
- SR Editor
- GSRL

Safety Analysis Tools
- FT Displayer
- Sim Displayer
- FT Plus

Safety Result Extraction

SAT-Repository
- SAT Management
- SAT-DB

Model Analysis
- ESM Generator

NuSMV-SA

R. Banach, School of Computer Science, University of Manchester, UK
M. Bozzano, Fondazione Bruno Kessler, FBK-IRST, Trento, Italy
The FSAP Architecture

Model Capturing
- Text Editor

FM Capturing
- FM Editor
- GFML

SR Capturing
- SR Editor
- GSRL

Safety Analysis Tools
- FT Displayer
- Sim Displayer
- FT Plus

Safety Result Extraction

SAT-Repository
- SAT Management
- SAT-DB

Model Analysis
- ESM Generator
- NuSMV-SA

NuSMV-SA
The FSAP Architecture

Model Capturing
- Text Editor

Safety Analysis Tools
- FT Displayer
- Sim Displayer
- FT Plus

Safety Result Extraction

FM Capturing
- FM Editor
- GFML

SR Capturing
- SR Editor
- GSRL

SAT-Repository
- SAT Management
- SAT-DB

Model Analysis
- ESM Generator

NuSMV-SA

R. Banach, School of Computer Science, University of Manchester, UK
M. Bozzano, Fondazione Bruno Kessler, FBK-IRST, Trento, Italy
A simple combinational digital circuit … nominal behaviour.

A1, A2, A3 are adders: eg. \( A1(\langle c2, c3, c5 \rangle) \equiv (c5 = c2 + c3) \)

\( F1, F2, F3 \) are fanouts: eg. \( F1(J1, \langle c1, c2 \rangle) \equiv (c1 = J1 \land c2 = J1) \)
Faulty behaviour … assumptions:

- Adders never fail.
- Fanouts have \textit{stuck\_at\_zero} faults at individual output signals.
- For any fanout, at most one output signal is faulty at any time.
FSAP Demo

Starting now …

For licensing, documentation, publications and more, visit:

http://sra.itc.it/tools/FSAP/

In order to fully mechanise the preceding, we need several things:

In order to fully mechanise the preceding, we need several things:

• A way of describing systems precisely —
• A way of describing system change/evolution precisely —
• A way of relating system change to an analysis of causes —
• A way of organising the information extracted into a Fault Tree —

In order to fully mechanise the preceding, we need several things:

- A way of describing systems precisely — state transition systems.
- A way of describing system change/evolution precisely — retrenchment.
- A way of relating system change to an analysis of causes — decomposing retrenchment simulation relation.
- A way of organising the information extracted into a Fault Tree — system structure and post-processing.
State Transition Systems

In general, state transition systems (with I/O) have:

- a state space,
- input and output spaces,
- labelled transitions ...
State Transition Systems

In general, state transition systems (with I/O) have:

- a state space,
- input and output spaces,
- labelled transitions ... eg. \( u - (i, Op, o) \rightarrow u' \) also written as \( Op(u, i, u', o) \)
State Transition Systems

In general, state transition systems (with I/O) have:

- a state space,
- input and output spaces,
- labelled transitions ... eg. \( u - (i, Op, o) \rightarrow u' \) also written as \( Op(u, i, u', o) \)

For combinational circuits, we can disregard the state, and simplify to:

\((i, Op, o)\) and \( Op(i, o) \)

Such relations are I/O transformers.

The key feature of these transformers is composition, which enables complex systems to be built out of simple components.
Composition of I/O transformers: parallel and (skew-)sequential composition.
Composition of I/O transformers: parallel and (skew-)sequential composition.

Parallel composition:

\( Op_1(i_1, o_1) \) and \\
\( Op_2(i_2, o_2) \) yield
Composition of I/O transformers: parallel and (skew-)sequential composition.

Parallel composition:
\[ Op_1(i_1, o_1) \text{ and } \]
\[ Op_2(i_2, o_2) \text{ yield } \]
\[ Op_{1|2} = Op_1 | Op_2 \]

\[ i_1 \quad Op_1 \quad o_1 \]
\[ i_2 \quad Op_2 \quad o_2 \]
Composition of I/O transformers: parallel and (skew-)sequential composition.

Parallel composition:

\[ \text{Op}_1(i_1, o_1) \text{ and Op}_2(i_2, o_2) \text{ yield Op}_{1|2} \equiv \text{Op}_1|\text{Op}_2 \]

or,

\[ \text{Op}_{1|2}((i_1, i_2), (o_1, o_2)) \equiv \text{Op}_1(i_1, o_1) \land \text{Op}_2(i_2, o_2) \]
Composition of I/O transformers: parallel and (skew-)sequential composition.

Parallel composition:

\[ Op_1(i_1, o_1) \text{ and } Op_2(i_2, o_2) \text{ yield } \]
\[ Op_{12} \equiv Op_1 | Op_2 \]

or,

\[ Op_{12}((i_1, i_2), (o_1, o_2)) \equiv Op_1(i_1, o_1) \land Op_2(i_2, o_2) \]

(Skew-)Sequential composition:

\[ Op_1((i_{11}, i_{12}), (o_{11}, o_{12})) \text{ and } Op_2((i_{21}, i_{22}), (o_{21}, o_{22})) \text{ yield } \]

\[ o_{11} \equiv i_{22} \]
Composition of I/O transformers: parallel and (skew-)sequential composition.

Parallel composition:
\[ Op_1(i_1, o_1) \text{ and } Op_2(i_2, o_2) \text{ yield } \]
\[ Op_{1\|2} \equiv Op_1 \parallel Op_2 \]
or,
\[ Op_{1\|2}((i_1, i_2), (o_1, o_2)) \equiv \]
\[ Op_1(i_1, o_1) \land Op_2(i_2, o_2) \]

(Skew-)Sequential composition:
\[ Op_1((i_{11}, i_{12}), (o_{11}, o_{12})) \text{ and } \]
\[ Op_2((i_{21}, i_{22}), (o_{21}, o_{22})) \text{ yield } \]
\[ Op_{1\triangleleft 2} \equiv Op_1 \delta Op_2 \text{ where } \]
\[ \delta \equiv (o_{11} = i_{22}) \]
Composition of I/O transformers: parallel and (skew-)sequential composition.

Parallel composition:
\[ Op_1(i_1, o_1) \text{ and } Op_2(i_2, o_2) \text{ yield } \]
\[ Op_{1|2} \equiv Op_1 \upharpoonright Op_2 \]
or,
\[ Op_{1|2}((i_1, i_2), (o_1, o_2)) \equiv Op_1(i_1, o_1) \land Op_2(i_2, o_2) \]

(Skew-)Sequential composition:
\[ Op_1((i_{11}, i_{12}), (o_{11}, o_{12})) \text{ and } \]
\[ Op_2((i_{21}, i_{22}), (o_{21}, o_{22})) \text{ yield } \]
\[ Op_{1;2} \equiv Op_1 \delta Op_2 \text{ where } \]
\[ \delta \equiv (o_{11} = i_{22}) \]
or,
\[ Op_{1;2}((i_{21}, i_{11}, i_{12}), (o_{21}, o_{22}, o_{12})) \equiv \]
\[ (\exists x \cdot Op_1((i_{11}, i_{12}), (x, o_{12})) \land Op_2((i_{21}, x), (o_{21}, o_{22}))) \]
System Evolution and Retrenchment

To accommodate the manifestation of faults in an otherwise working system, in a way that permits formal/mechanical analysis and processing, we need a formal way of describing system evolution.

Most formal techniques (that deal with relationships between system models) are geared to the preservation of properties. Typically these are notions of refinement.

Retrenchment is distinctive in that it is geared to capturing differences between system models, irrespective of properties preserved, or not.

Yet, retrenchment seeks to be as compatible with notions of refinement as is practical, in order to get the best of both worlds.

The I/O transformer world simplifies things a lot.
System Evolution and Retrenchment

Retrenchment relates two systems where they have points of contact:

- Identifying operations —
- Identifying scope of common operations’ relationship —
- Identifying properties of simulating behaviour —
- Identifying properties of divergent behaviour —
System Evolution and Retrenchment

Retrenchment relates two systems where they have points of contact:

- Identifying operations — most easily done via name identity.
- Identifying scope of common operations’ relationship — by restricting simulation.
- Identifying properties of simulating behaviour — by specifying simulation.
- Identifying properties of divergent behaviour — by specifying exceptions.
System Evolution and Retrenchment

Retrenchment relates two systems where they have points of contact:

- Identifying operations — most easily done via name identity.
- Identifying scope of common operations’ relationship — by restricting simulation.
- Identifying properties of simulating behaviour — by specifying simulation.
- Identifying properties of divergent behaviour — by specifying exceptions.
- [ Relating state spaces of the two systems — via a retrieve relation. ]

Retrenchment typically relates an abstract system \textit{Abs} and a concrete system \textit{Conc}, using a simulation-like criterion to connect smoothly with refinement-like behaviour, (the latter is of less interest here).
Suppose we have *Abs* and *Conc*, both with an operation *Op*, and where the abstract system’s inputs and outputs are *i* and *o*, and the concrete system’s inputs and outputs are *j* and *p*. A retrenchment from *Abs* to *Conc* centres round a proof obligation:

As with I/O transformers, composition is a crucial issue.
Composition of retrenchments: parallel and (skew-)sequential composition.

Parallel composition:

\[ Op_1(i_1, o_1) \text{ and } Op_2(i_2, o_2) \text{ yield } Op_{1|2} \equiv Op_1 | Op_2 \]

or,

\[ Op_{1|2}((i_1, i_2), (o_1, o_2)) \equiv Op_1(i_1, o_1) \land Op_2(i_2, o_2) \]
Composition of retrenchments: parallel and (skew-)sequential composition.

Parallel composition:

\[ Op_1(i_1, o_1) \text{ and } Op_2(i_2, o_2) \text{ yield } Op_{1|2} \equiv Op_1 | Op_2 \]

or,

\[ Op_{1|2}((i_1, i_2), (o_1, o_2)) \equiv Op_1(i_1, o_1) \land Op_2(i_2, o_2) \]

Let \( Op_{1,A}(i_1, o_1) \) be retrenched to \( Op_{1,C}(j_1, p_1) \) via

\[ W_{Op,1}(i_1, j_1), O_{Op,1}(o_1, p_1, i_1, j_1), C_{Op,1}(o_1, p_1, i_1, j_1) \]

Let \( Op_{2,A}(i_2, o_2) \) be retrenched to \( Op_{2,C}(j_2, p_2) \) via

\[ W_{Op,2}(i_2, j_2), O_{Op,2}(o_2, p_2, i_2, j_2), C_{Op,2}(o_2, p_2, i_2, j_2) \]
Composition of retrenchments: parallel and (skew-)sequential composition.

Parallel composition:
\[ Op_1(i_1, o_1) \text{ and } Op_2(i_2, o_2) \text{ yield } Op_{1|2} \equiv Op_1|Op_2 \]
or,
\[ Op_{1|2}((i_1, i_2), (o_1, o_2)) \equiv Op_1(i_1, o_1) \land Op_2(i_2, o_2) \]

Let \( Op_{1,A}(i_1, o_1) \) be retrenched to \( Op_{1,C}(j_1, p_1) \) via
\[ W_{Op,1}(i_1, j_1) , O_{Op,1}(o_1, p_1, i_1, j_1) , C_{Op,1}(o_1, p_1, i_1, j_1) \]

Let \( Op_{2,A}(i_2, o_2) \) be retrenched to \( Op_{2,C}(j_2, p_2) \) via
\[ W_{Op,2}(i_2, j_2) , O_{Op,2}(o_2, p_2, i_2, j_2) , C_{Op,2}(o_2, p_2, i_2, j_2) \]

Then \( Op_{1|2,A} \) is retrenched to \( Op_{1|2,C} \) via
\[ W_{Op,1|2} \equiv W_{Op,1}|W_{Op,2} \]
\[ O_{Op,1|2} \equiv O_{Op,1}|O_{Op,2} \]
\[ C_{Op,1|2} \equiv O_{Op,1}|C_{Op,2} \lor C_{Op,1}|O_{Op,2} \lor C_{Op,1}|C_{Op,2} \]
Composition of retrenchments: parallel and (skew-)sequential composition.

(Skew-)Sequential composition:
\[ Op_1(\langle i_{11}, i_{12} \rangle, \langle o_{11}, o_{12} \rangle) \quad \text{and} \quad Op_2(\langle i_{21}, i_{22} \rangle, \langle o_{21}, o_{22} \rangle) \]
\[ \text{yield} \quad Op_{1 \delta 2} \equiv Op_1 \delta Op_2 \]
where \[ \delta \equiv (o_{11} = i_{22}) \]
or,
\[ Op_{1 \delta 2}(\langle i_{21}, i_{11}, i_{12} \rangle, \langle o_{21}, o_{22}, o_{12} \rangle) \equiv \]
\[ (\exists x \cdot Op_1(\langle i_{11}, i_{12} \rangle, \langle x, o_{12} \rangle) \wedge Op_2(\langle i_{21}, x \rangle, \langle o_{21}, o_{22} \rangle)) \]
Composition of of retrenchments: parallel and (skew-)sequential composition.

(Skew-)Sequential composition:

\[ Op_1(\langle i_{11}, i_{12} \rangle, \langle o_{11}, o_{12} \rangle) \quad \text{and} \quad \]
\[ Op_2(\langle i_{21}, i_{22} \rangle, \langle o_{21}, o_{22} \rangle) \quad \text{yield} \]
\[ Op_{1 \circ 2} \equiv Op_1 \delta Op_2 \quad \text{where} \]
\[ \delta \equiv (o_{11} = i_{22}) \]

or,

\[ Op_{1 \circ 2}(\langle i_{21}, i_{11}, i_{12} \rangle, \langle o_{21}, o_{22}, o_{12} \rangle) \equiv \]
\[ (\exists x \cdot Op_1(\langle i_{11}, i_{12} \rangle, \langle x, o_{12} \rangle)) \land Op_2(\langle i_{21}, x \rangle, \langle o_{21}, o_{22} \rangle) \]

Let \( Op_{1,A}(\langle i_{11}, i_{12} \rangle, \langle o_{11}, o_{12} \rangle) \) be retrenched to \( Op_{1,C}(\langle j_{11}, j_{12} \rangle, \langle p_{11}, p_{12} \rangle) \) via
\[ W_{Op,1}(\langle i_{11}, i_{12} \rangle, \langle j_{11}, j_{12} \rangle), \quad Op_{Op,1}(\langle o_{11}, o_{12} \rangle, \langle p_{11}, p_{12} \rangle, \langle i_{11}, i_{12} \rangle, \langle j_{11}, j_{12} \rangle), \]
\[ C_{Op,1}(\langle o_{11}, o_{12} \rangle, \langle p_{11}, p_{12} \rangle, \langle i_{11}, i_{12} \rangle, \langle j_{11}, j_{12} \rangle) \]

Let \( Op_{2,A}(\langle i_{21}, i_{22} \rangle, \langle o_{21}, o_{22} \rangle) \) be retrenched to \( Op_{2,C}(\langle j_{21}, j_{22} \rangle, \langle p_{21}, p_{22} \rangle) \) via
\[ W_{Op,2}(\langle i_{21}, i_{22} \rangle, \langle j_{21}, j_{22} \rangle), \quad Op_{Op,2}(\langle o_{21}, o_{22} \rangle, \langle p_{21}, p_{22} \rangle, \langle i_{21}, i_{22} \rangle, \langle j_{21}, j_{22} \rangle), \]
\[ C_{Op,2}(\langle o_{21}, o_{22} \rangle, \langle p_{21}, p_{22} \rangle, \langle i_{21}, i_{22} \rangle, \langle j_{21}, j_{22} \rangle) \]
We make some simplifying assumptions:

The within relation splits on pairs of signals:

\[
W_{Op,1}(\langle i_{11}, i_{12} \rangle, \langle j_{11}, j_{12} \rangle) \equiv W_{Op,1}^1(i_{11}, j_{11}) \land W_{Op,2}^2(i_{12}, j_{12})
\]
\[
W_{Op,2}(\langle i_{21}, i_{22} \rangle, \langle j_{21}, j_{22} \rangle) \equiv W_{Op,1}^1(i_{21}, j_{21}) \land W_{Op,2}^2(i_{22}, j_{22})
\]

The results of the first step fall into the within relation of the second step:

\[
(\forall a, c \cdot (O_{Op,1}(\langle a, o_{12} \rangle, \langle c, p_{12} \rangle, \langle i_{11}, i_{12} \rangle, \langle j_{11}, j_{12} \rangle) \lor C_{Op,1}(\langle a, o_{12} \rangle, \langle c, p_{12} \rangle, \langle i_{11}, i_{12} \rangle, \langle j_{11}, j_{12} \rangle)) \implies W_{Op,2}(a, c)
\]
We make some simplifying assumptions:

The within relation splits on pairs of signals:

\[ W_{Op,1}(\langle i_{11}, i_{12}, j_{11}, j_{12} \rangle) \equiv W_{Op,1}^{1}(i_{11}, j_{11}) \land W_{Op,1}^{2}(i_{12}, j_{12}) \]
\[ W_{Op,2}(\langle i_{21}, i_{22}, j_{21}, j_{22} \rangle) \equiv W_{Op,2}^{1}(i_{21}, j_{21}) \land W_{Op,2}^{2}(i_{22}, j_{22}) \]

The results of the first step fall into the within relation of the second step:

\[ (\forall a, c \bullet (O_{Op,1}(\langle a, o_{12}, c, p_{12}, i_{11}, i_{12}, j_{11}, j_{12} \rangle) \lor C_{Op,1}(\langle a, o_{12}, c, p_{12}, i_{11}, i_{12}, j_{11}, j_{12} \rangle)) \Rightarrow W_{Op,2}(a, c)) \]

Then \( Op_{1,2,A} \) is retrenched to \( Op_{1,2,C} \) via

\[ W_{Op,1}(a_{2}) \equiv W_{Op,2}(i_{21}, j_{21}) \land W_{Op,1}(i_{11}, j_{11}) \land W_{Op,1}(i_{12}, j_{12}) \]
\[ O_{Op,1}(a_{2}) \equiv O_{Op,1}(a_{2}) \land O_{Op,2}(a_{2}) \equiv (\exists a, c \bullet O_{Op,1}(\langle a, o_{12}, c, p_{12}, i_{11}, i_{12}, j_{11}, j_{12} \rangle) \land O_{Op,2}(\langle o_{21}, o_{22}, p_{21}, p_{22}, i_{21}, a, j_{21}, c \rangle)) \]
\[ C_{Op,1}(a_{2}) \equiv O_{Op,1}(a_{2}) \land C_{Op,2}(a_{2}) \lor C_{Op,1}(a_{2}) \land C_{Op,2}(a_{2}) \]
Decomposing the Retrenchment Simulation Relation

For FT analysis we focus on the retrenchment simulation relation:

\[
\Sigma^1 \equiv W_{Op}(i, j) \land O_{PC}(j, p) \land O_{PA}(i, o) \land (O_{Op}(o, p, i, j) \lor C_{Op}(o, p, i, j))
\]

‘everything round the square is true’
Decomposing the Retrenchment Simulation Relation

For FT analysis we focus on the retrenchment simulation relation:

\[ \Sigma^1 \equiv W_{Op}(i, j) \land O_{PC}(j, p) \land O_{PA}(i, o) \land (O_{Op}(o, p, i, j) \lor C_{Op}(o, p, i, j)) \]

In applying retrenchment to FT generation via fault injection controlled by fault variables (for digital circuits), we can make further simplifying assumptions:

1. *Abs* and *Conc* correspond to nominal and faulty systems respectively.
2. Because of 1, corresponding data types in *Abs* and *Conc* are the same.
3. Because of 2, we can view *Abs* as embedded in *Conc* and so dispense with *Abs*.
4. Since *O_{Op}* and *C_{Op}* are design choices, we can embed *O_{PA}, O_{PC}* in *O_{Op}, C_{Op}*. 
5. Because of 4, we can take *W_{Op}* as true.
Decomposing the Retrenchment Simulation Relation

By the previous assertions, the simulation relation for an operation reduces to:

\[ \Sigma^1 \equiv O_{Op}(p, \hat{j}) \lor C_{Op}(p, \hat{j}) \]

Here:

1. We express Abs behaviour in terms of Conc variables in \( O_{Op} \) and \( C_{Op} \).
2. We distinguish Abs behaviour from Conc behaviour in \( O_{Op} \) and \( C_{Op} \) via the truth/falsehood of \textbf{fault variables}.
   - So ‘all fault variables false’ means nominal behaviour, belonging to \( O_{Op} \);
   - and ‘at least one fault variable true’ means faulty behaviour, belonging to \( C_{Op} \).

We also assume all components are \textbf{input enabled} (i.e. for every input there is \textit{some} output, though not necessarily \textit{vice versa}).
5. FT Extraction for Combinational Circuits.

Time for an example:

A simple combinational digital circuit ... nominal behaviour.

\( A1, A2, A3 \) are adders: eg. \( A1(\langle c2, c3 \rangle, c5) \equiv (c5 = c2 + c3) \)

\( F1, F2, F3 \) are fanouts: eg. \( F1(J1, \langle c1, c2 \rangle) \equiv (c1 = J1 \land c2 = J1) \)

\( K0, K1, K2 \), etc. define the structure via parallel and (skew-)sequential composition.
5. FT Extraction for Combinational Circuits.

Faulty behaviour ... assumptions:

• Adders never fail.

• Fanouts have stuck\_at\_zero faults on individual output signals. For any fanout, at most one output signal is faulty at any time.

We can easily express this using retrenchments.

Adders:

\[
\begin{align*}
W_{A1}(\langle c_2, c_3 \rangle) & \equiv \text{true} \\
O_{A1}(c_5, \langle c_2, c_3 \rangle) & \equiv (c_5 = c_2 + c_3) \\
C_{A1}(c_5, \langle c_2, c_3 \rangle) & \equiv \text{false}
\end{align*}
\]

(Only nominal behaviour.)
Fanouts:

\[ W_{F1}(J1) \equiv true \]
\[ O_{F1}(\langle c1,c2 \rangle, J1) \equiv (c1 = c2 = J1) \]
\[ C_{F1}(\langle c1,c2 \rangle, J1) \equiv (F1.c1.0 \land c1 = 0 \land c2 = J1) \oplus \\
(F1.c2.0 \land c1 = J1 \land c2 = 0) \]

Boolean fault variables

(Faulty behaviour captured using fault variables \( F1.c1.0 \) and \( F1.c2.0 \) in the concession. When true, they ‘switch on’ the relevant fault.)
Fanouts:

\[ W_{F1}(J1) \equiv \text{true} \]
\[ O_{F1}([c1,c2], J1) \equiv (c1 = c2 = J1) \]
\[ C_{F1}([c1,c2], J1) \equiv (F1.c1.0 \land c1 = 0 \land c2 = J1) \oplus (F1.c2.0 \land c1 = J1 \land c2 = 0) \]

Boolean fault variables

‘exclusive OR’

(Faulty behaviour captured using fault variables \(F1.c1.0\) and \(F1.c2.0\) in the concession. When true, they ‘switch on’ the relevant fault.)

So now that we have nominal and faulty behaviours for all the components, all we need to do is to compose them all together, and analyse the result. Right?
Fanouts:

\[ W_{F1}(J1) \equiv true \]
\[ O_{F1}((c1, c2), J1) \equiv (c1 = c2 = J1) \]
\[ C_{F1}((c1, c2), J1) \equiv (F1.c1.0 \land c1 = 0 \land c2 = J1) \oplus (F1.c2.0 \land c1 = J1 \land c2 = 0) \]

Boolean fault variables

(Faulty behaviour captured using fault variables \( F1.c1.0 \) and \( F1.c2.0 \) in the concession. When true, they ‘switch on’ the relevant fault.)

So now that we have nominal and faulty behaviours for all the components, all we need to do is to compose them all together, and analyse the result. Right? Wrong!

By composing everything together, we create a monolith that may be too complex. Using the soundness of retrenchment composition, we decompose lazily, breaking the task down, and extracting only what we need.
Retrenchment Directed Analysis of a TLE

Define a TLE to be an expression in the external variables of the system.

Given a TLE, we can use the structure of the system to guide the decomposition and instantiation of the simulation relation. Assuming the TLE indeed represents a fault, we decompose the concession to derive a resolution tree (RT) for the TLE.

Eg. TLE: $J_1 = J_2 = P_1 = 1$ (with $P_2$ regarded as irrelevant)

Since we look for causes of the outputs, we search backwards from the outputs.
The TLE.

TLE: \( \exists P_2 \cdot P_1 = 1 \land J_1 = 1 \land J_2 = 1 \ldots \)
Analyse $P1 = 1$.
Two causes.

$$C_{K0} \equiv O_{K2} \odot C_{K1} \lor C_{K2} \odot O_{K1} \lor C_{K2} \odot C_{K1}$$

$$C_{K1} \equiv O_{A2} | C_{A3} \lor C_{A2} | O_{A3} \lor C_{A2} | C_{A3}$$
Left hand cause.
Trace through $F_3$, and through $A_1$.
Two causes.

$C_{K2} \equiv O_{F_3} \circ C_{K3} \lor C_{F_3} \circ O_{K3} \lor C_{F_3} \circ C_{K3}$
$C_{K3} \equiv O_{A1} \circ C_{K4} \lor C_{A1} \circ O_{K4} \lor C_{A1} \circ C_{K4}$
Left hand cause.
Unsatisfiable.
Requires both
\( F_1.c1.0 \) and \( F_1.c2.0 \).
Backtrack.

\[ C_{K4} \equiv O_{F1} | C_{F2} \vee C_{F1} | O_{F2} \vee C_{F1} | C_{F2} \]
Right hand cause.
Satisfiable.
A basic cause
of the TLE.

\[ C_{K4} \equiv O_{F1} \lor C_{F2} \lor C_{F1} \lor O_{F2} \lor C_{F1} \lor C_{F2} \]
... eventually ...
The full RT of the example TLE (compressed).
Minimisation

Naive RT generation can yield many redundant, non-minimal, causes of the TLE. We can use semantic/algebraic knowledge to prune the RT as it’s built. Eg.

- Discarding non-needed subtrees; (from generated faults without dataflow to TLE).
- Discarding subtrees at input-insensitive faults; (eg. stuck-at faults).
- Discarding locally subsumed expressions.

However ... need a final subsumption check.

This gives a rapid path to the construction of the *Minimal Cut Sets* of the TLE.
TLE: $\left( \exists P_2 \cdot P_1 = 1 \land J_1 = 1 \land J_2 = 1 \ldots \right)$

Redundant
Generating a Fault Tree

The resolution tree can be used to derive a FT according to standard FT conventions.

- Basic Fault nodes.
- Intermediate events.

Doing this on the running example yields:

\[
\text{TLE: } (\exists P2 \cdot P1 = 1 \land J1 = 1 \land J2 = 1 \ldots)
\]

\[
\begin{align*}
c1,c5,c6 &= 0,1,1 \\
c1,c6 &= 1,0 \\
c1,c5,c6 &= 1,0,0
\end{align*}
\]

- \(c2 = 1\) \(F1.c1\)
- \(c3 = 0\) \(F2.c3\)
- \(c2 = 0\) \(F1.c2\)
- \(c3 = 0\) \(F2.c3\)

- \(F3.c6\)

R. Banach, School of Computer Science, University of Manchester, UK
M. Bozzano, Fondazione Bruno Kessler, FBK-IRST, Tento, Italy

We have a brief look at the theoretical framework that underpins the preceding.

We have a brief look at the theoretical framework that underpins the preceding.

- Standardise variables: external variables; internal variables, fault variables.
- Standardise expressions: system definition $S(J, P)$, TLE.
- Standardise system structure language.
- Standardise terms: fault config., cut set, etc.
Resolution Tree Algorithm

An abstract version of what a real algorithm would do. Features unrealistic elements, eg. angelic non-determinism.

Four top level cases for TLE and system:

(a) $O_S$ unsatisfiable; $C_S$ unsatisfiable
(b) $O_S$ satisfiable; $C_S$ unsatisfiable

\[\begin{array}{c}
\text{TLE} \leftarrow \text{TLE node} \\
\text{TLE} \\
\text{GOAS} \\
\text{Global } O_S \\
\text{ASsignment node}
\end{array}\]
Four top level cases for TLE and system:

(c) $O_S$ unsatisfiable; $C_S$ satisfiable

(d) $O_S$ satisfiable; $C_S$ satisfiable

- angelic ASsiGnment to interface variables node

- Concession node
Each Concession node is recursively expanded:

\[ C \equiv BF \]

becomes

\[ C \equiv O_1|C_2 \lor C_1|O_2 \lor C_1|C_2 \]

becomes

\[ O \text{ ASignment node} \]
Each Concession node is recursively expanded:

\[ C \equiv O_1 \circ C_2 \lor O_2 \lor C_1 \circ C_2 \]

\[ C \] becomes

- Angelic Assignment to interface variables node
Properties of the Analysis

- It terminates.
- Each maximal consistent cut set (i.e. all children of an AND, one child of an OR) assigns to all variables (when negations of all absent fault variables are added).
- It is sound (i.e. every derivation produced, yields a valid cause of the TLE).
- It is complete (i.e. every valid cause of the TLE has a derivation in the system).
Properties of the Analysis

- It terminates.
- Each maximal consistent cut set (i.e. all children of an AND, one child of an OR) assigns to all variables (when negations of all absent fault variables are added).
- It is sound (i.e. every derivation produced, yields a valid cause of the TLE).
- It is complete (i.e. every valid cause of the TLE has a derivation in the system).

Proved by induction.
Properties of the Analysis

• It terminates.
• Each maximal consistent cut set (i.e. all children of an AND, one child of an OR) assigns to all variables (when negations of all absent fault variables are added).
• It is sound (i.e. every derivation produced, yields a valid cause of the TLE).
• It is complete (i.e. every valid cause of the TLE has a derivation in the system).

Proved by induction.

N.B. The algorithm works top-down; the proof works bottom-up. The theory thus gives good support for the algorithmic approach.
From Resolution Trees to Fault Trees

To convert the resolution trees produced by the algorithm to something resembling conventional fault trees, we must:

- Eliminate OAS nodes (short-circuit).
- Eliminate redundant logical connectives (ones with a single child — short-circuit).
- Compress chains of ASG nodes.
- Insert IE nodes (intermediate event) between adjacent connectives.
From Resolution Trees to Fault Trees

To convert the resolution trees produced by the algorithm to something resembling conventional fault trees, we must:

• Eliminate OAS nodes (short-circuit).
• Eliminate redundant logical connectives (ones with a single child — short-circuit).
• Compress chains of ASG nodes.
• Insert IE nodes (intermediate event) between adjacent connectives.

Straightforward enough to do algorithmically.
Example

...
Example

```
OAS: v = 0 ...

OR

ASG: x = 5

AND

ASG: y = 3

OR

ASG: z = 2

ASG: z = 5
```

...
Example

OAS: \( v = 0 \) …

OR

ASG: \( x = 5 \)

AND

ASG: \( y = 3 \)

OR

ASG: \( z = 2 \)

ASG: \( z = 5 \)

ASG: \( z = 5 \)

ASG: \( x = 5 \)

AND

ASG: \( y = 3 \)

OR

ASG: \( z = 2 \)

ASG: \( z = 5 \)

…
Example

\[ \text{OAS: } v = 0 \ldots \]

\[ \text{OR} \]

\[ \text{ASG: } x = 5 \]

\[ \text{AND} \]

\[ \text{ASG: } y = 3 \]

\[ \text{OR} \]

\[ \text{ASG: } z = 2 \]
\[ \text{AND} \]

\[ \text{ASG: } y = 3 \]

\[ \text{OR} \]

\[ \text{ASG: } z = 2 \]
\[ \text{ASG: } z = 5 \]

\[ \ldots \]

\[ \ldots \]
Example

OAS: \( v = 0 \) …

OR

ASG: \( x = 5 \)

AND

ASG: \( y = 3 \)

OR

ASG: \( z = 2 \)  ASG: \( z = 5 \)

…  …

ASG: \( z = 2 \)  ASG: \( z = 5 \)

…  …

IE: \( z = 2 \lor z = 5 \)
7. Clocked Acyclic Circuits, Causal Relations, Retrenchments.

We want extend the combinational circuit analysis to clocked (acyclic) circuits.

• Clocked circuits have state.

• State itself is no problem for retrenchment, but has a notable disadvantage: it doesn’t disappear when you compose subsystems — it just accumulates.

• Problematic for a hierarchical approach — you want lower level detail to disappear.
7. Clocked Acyclic Circuits, Causal Relations, Retrenchments.

We want extend the combinational circuit analysis to clocked (acyclic) circuits.

- Clocked circuits have state.
- State itself is no problem for retrenchment, but has a notable disadvantage: it doesn’t disappear when you compose subsystems — it just accumulates.
- Problematic for a hierarchical approach — you want lower level detail to disappear.

Solution: translate the effects of state into I/O dependencies.
Advantage: can reuse combinational theory more or less verbatim.
Clocked Acyclic Circuits and Causal Relations

A clocked circuit: [Diagram showing two paths with delays]

Some facts for synchronous clocked circuits:

- A given input signal and output signal may be linked by more than one path, with different delays, because of different stateful elements along the way.

- An output value is caused by the input values at all the paths that can reach it; input and output values may be deterministically or nondeterministically related.

- Different output values may be non-trivially correlated (cf. fanout faults above); regardless of (non)determinism.

This leads to an I/O formulation of clocked circuits in terms of causal relations.
Clocked Acyclic Circuits and Causal Relations

Suppose we have a synchronous clocked circuit \( AcyOp \):

Suppose for \( AcyOp \) we have:

1. The I/O paths and delays are as indicated.
2. Outputs \( o_1 \) and \( o_2 \) are non-trivially correlated.
3. Output value of \( o_2 \) is required one time step later than \( o_1 \) (from the same inputs).
4. Output \( o_0 \) is independent of \( o_1 \) and \( o_2 \).
5. Input \( i_0 \) is not used.
Clocked Acyclic Circuits and Causal Relations

Then circuit $AcyOp$ will have causal relation $AcyOpS$:

\[
\begin{align*}
&AcyOpS \\
&\text{ where, in } AcyOpS: \\
&1. \text{ Output values occur once only in the output signature of } AcyOpS. \\
&2. \lambda^0 \text{ and } \lambda^{12} \text{ are unrelated, and refer to independence of } o_0 \text{ from } o_1 \text{ and } o_2. \\
&3. \text{ Outputs } o_1(t + \lambda^{12}) \text{ and } o_2(t + \lambda^{12} + 1) \text{ refer to times } o_1 \text{ and } o_2 \text{ are delivered.} \\
&4. \text{ Input signature of } AcyOpS \text{ is governed by the set of input values needed to produce the required output values ... i.e.} \\
&\quad i_1(t + \lambda^0) \text{ is needed to produce } o_0(t + \lambda^0), \\
&\quad i_1(t + \lambda^{12} - 1), i_2(t + \lambda^{12} - 2), i_2(t + \lambda^{12} - 3) \text{ produce } o_1(t + \lambda^{12}) \text{ and } o_2(t + \lambda^{12} + 1).
\end{align*}
\]
Clocked Acyclic Circuits and Causal Relations

Then circuit $AcyOp$ will have causal relation $AcyOpS$:

In logical terms $AcyOpS$ is given by:

\[
AcyOpS(\langle i_1(t_{\lambda^0}), i_1(t_{\lambda^{12}} -1), i_2(t_{\lambda^{12}} -2), i_2(t_{\lambda^{12}} -3) \rangle, \langle o_0(t_{\lambda^0}), o_1(t_{\lambda^{12}}), o_2(t_{\lambda^{12}} + 1) \rangle) \equiv
\]

\[
AcyOpS^0(i_1(t_{\lambda^0}), o_0(t_{\lambda^0})) \wedge
\]

\[
AcyOpS^{12}(\langle i_1(t_{\lambda^{12}} -1), i_2(t_{\lambda^{12}} -2), i_2(t_{\lambda^{12}} -3) \rangle, \langle o_1(t_{\lambda^{12}}), o_2(t_{\lambda^{12}} + 1) \rangle) \wedge
\]

\[
(\lambda^0 = \lambda^{12} - 1 \Rightarrow i_1(t_{\lambda^0}) = i_1(t_{\lambda^{12}} - 1))
\]

$AcyOpS$ thus decomposes into two irreducible subrelations $AcyOpS^0$ and $AcyOpS^{12}$. 
The general picture:

- The circuit will determine the irreducible subrelations of the causal relation; (in general, components with ‘more than one output’ will introduce correlations).

- Conversely, any causal relation can be realised by combinational circuits, fanouts, and delays, eg. $AcyOpS$ above:
The general picture:

- The circuit will determine the irreducible subrelations of the causal relation; (in general, components with 'more than one output' will introduce correlations).

- Conversely, any causal relation can be realised by combinational circuits, fanouts, and delays, eg. AcyOpS above:

  \[
  \text{AcyOpS}^{\gamma}((i_r(t_{\lambda \gamma} - \theta_{r,1}), i_r(t_{\lambda \gamma} - \theta_{r,2}), i_s(t_{\lambda \gamma} - \theta_s), \ldots), (o_u(t_{\lambda \gamma} + \phi_u), o_w(t_{\lambda \gamma} + \phi_w), \ldots))
  \]

  - set of outputs
  - delays for input \(i_r\) relative to \(t_{\lambda \gamma}\)
  - delay for output \(o_u\) relative to \(t_{\lambda \gamma}\)

  - fanout element
  - delay element
  - combinational circuits
Expressing everything in terms of causal relations puts all timing info into the names of the variables.

Eg. we regard $o_u(t_{k\gamma} + \phi_u)$ say, as the name of a variable $'o_u(t_{k\gamma} + \phi_u)'$ ... NOT as variable $o_u$ evaluated at $(t_{k\gamma} + \phi_u)$, and subsequently supplying a value for the output stream named $o_u$. 
Expressing everything in terms of causal relations puts all timing info into the names of the variables.

Eg. we regard $o_u(t_{\lambda\gamma} + \phi_u)$ say, as the name of a variable $o_u(t_{\lambda\gamma} + \phi_u)$ ... NOT as variable $o_u$ evaluated at $(t_{\lambda\gamma} + \phi_u)$, and subsequently supplying a value for the output stream named $o_u$.

So, acyclic synchronous timed circuits are represented using machinery for combinational circuits ... just usually many more variables around, that's all.
Expressing everything in terms of causal relations puts all timing info into the names of the variables.

Eg. we regard \( o_u(t_{\lambda^\gamma} + \phi_u) \) say, as the name of a variable ‘\( o_u(t_{\lambda^\gamma} + \phi_u) \)’ ... NOT as variable \( o_u \) evaluated at \( (t_{\lambda^\gamma} + \phi_u) \), and subsequently supplying a value for the output stream named \( o_u \).

So, acyclic synchronous timed circuits are represented using machinery for combinational circuits ... just usually many more variables around, that’s all.

One exception ... need to ensure appropriate constraints like

\[
(\lambda^0 = \lambda^{12} - 1 \Rightarrow i_1(t_{\lambda^0}) = i_1(t_{\lambda^{12} - 1}))
\]

are included, when the same input stream is used for independent outputs.
Expressing everything in terms of causal relations puts all timing info into the names of the variables.

Eg. we regard $o_u(t_{\lambda\gamma} + \phi_u)$ say, as the name of a variable ‘$o_u(t_{\lambda\gamma} + \phi_u)$’ ... NOT as variable $o_u$ evaluated at $(t_{\lambda\gamma} + \phi_u)$, and subsequently supplying a value for the output stream named $o_u$.

So, acyclic synchronous timed circuits are represented using machinery for combinational circuits ... just usually many more variables around, that’s all.

One exception ... need to ensure appropriate constraints like

\[(\lambda^0 = \lambda^{12} - 1 \Rightarrow i_1(t_{\lambda}^0) = i_1(t_{\lambda}^{12} - 1))\]

are included, when the same input stream is used for independent outputs.

Now ... composition properties follow as previously (with above proviso):

Parallel and (skew-)sequential composition of causal relations for synchronous acyclic clocked circuits.
Example ... skew-sequential composition of circuits and their causal relations:

\[ \begin{align*}
    i_{10} & \quad o_{20} \\
    i_{11} & \quad o_{21} \\
    i_{12} & \quad o_{22} \\
\end{align*} \]

AcyOp_1  AcyOp_2

\[ \begin{align*}
    i_{11}(t + \lambda_2^0) & \quad o_{20}(t + \lambda_2^0) \\
    i_{11}(t + \lambda_2^{12} - 1) & \quad o_{21}(t + \lambda_2^{12}) \\
    i_{11}(t + \lambda_2^{12} - 3) & \quad o_{22}(t + \lambda_2^{12} + 1) \\
    i_{12}(t + \lambda_2^{12} - 4) & \quad o_{12}(t + \lambda_2^{12} - 1) \\
    i_{12}(t + \lambda_2^{12} - 5) & \quad o_{12}(t + \lambda_2^{12} - 2) \\
\end{align*} \]

\[ \begin{align*}
    i_{11}(t + \lambda_2^{12} - 4) & \\
    i_{12}(t + \lambda_2^{12} - 5) & \\
    i_{12}(t + \lambda_2^{12} - 6) & \\
\end{align*} \]

\( \lambda \)-delays for AcyOpS_1 derived from \( \lambda \)-delays for AcyOpS_2

(utilises fact that \( \lambda \)'s are free parameters, and that circuits are time invariant)

Undoubtedly somewhat complicated due to the large number of variables, but no surprises in principle.
Strictly speaking, there’s nothing to do. *Time-as-part-of-variable-name* convention reduces retrenchment to the combinational case.

Retrenchment simulation relation for $AcyOpS^0(i_1(t_{\lambda^0}), o_0(t_{\lambda^0}))$ example:

$$\Sigma^1 \equiv W_{AcyOpS^0}(i_1(t_{\lambda^0}), j_1(t_{\lambda^0})) \land AcyOpS^0_C(j_1(t_{\lambda^0}), p_0(t_{\lambda^0})) \land AcyOpS^0_A(i_1(t_{\lambda^0}), o_0(t_{\lambda^0})) \land$$

$$(O_{AcyOpS^0}(o_0(t_{\lambda^0}), p_0(t_{\lambda^0}), i_1(t_{\lambda^0}), j_1(t_{\lambda^0})) \lor C_{AcyOpS^0}(o_0(t_{\lambda^0}), p_0(t_{\lambda^0}), i_1(t_{\lambda^0}), j_1(t_{\lambda^0})))$$

Messier variable names, but otherwise as before.

So, parallel and (skew-)sequential composition of retrenchments as previously.
8. FT Extraction for Clocked Acyclic Circuits.

Modify the earlier example to introduce time aspects:

Nominal behaviour ... (suppressing $\lambda$’s).

$A1$ ... are instantaneous adders: eg. $A1(\langle c2(t), c3(t) \rangle, c5(t)) \equiv (c5(t) = c2(t) + c3(t))$

$F1$ ... are delay-1 fanouts: eg. $F1(J1(t-1), \langle c1(t), c2(t) \rangle) \equiv (c1(t) = J1(t-1) \land c2(t) = J1(t-1))$

$K0$ ... define the structure via parallel and (skew-)sequential composition.
8. FT Extraction for Clocked Acyclic Circuits.

Faulty behaviour ... assumptions ..................... just as before:

- Adders never fail.
- Fanouts have **stuck_at_zero** faults on individual output signals. For any fanout, at most one output signal is faulty at any time.

Using retrenchments:

Adders:

$$W_{A1}(\langle c2(t), c3(t) \rangle) \equiv \text{true}$$
$$O_{A1}(c5(t), \langle c2(t), c3(t) \rangle) \equiv (c5(t) = c2(t) + c3(t))$$
$$C_{A1}(c5(t), \langle c2(t), c3(t) \rangle) \equiv \text{false}$$

(Only nominal behaviour.)
Fanouts:

\[ \begin{align*}
W_{F1}(J1(t - 1)) & \equiv \text{true} \\
O_{F1}(\langle c1(t), c2(t) \rangle, J1(t - 1)) & \equiv (c1(t) = c2(t) = J1(t - 1)) \\
C_{F1}(\langle c1(t), c2(t) \rangle, J1(t - 1)) & \equiv (F1.c1.0(t) \land c1(t) = 0 \land c2(t) = J1(t - 1)) \oplus (F1.c2.0(t) \land c1(t) = J1(t - 1) \land c2(t) = 0) 
\end{align*} \]

*Time dependent* Boolean fault variables

(Time dependent fault variables \( F1.c1.0(t) \) and \( F1.c2.0(t) \) in the concession ‘switch on’ the relevant fault.)
Fanouts:

\[
W_{F_1}(J_1(t-1)) \equiv \text{true} \\
O_{F_1}(\langle c_1(t), c_2(t) \rangle, J_1(t-1)) \equiv (c_1(t) = c_2(t) = J_1(t-1)) \\
C_{F_1}(\langle c_1(t), c_2(t) \rangle, J_1(t-1)) \equiv (F_1.c_1.0(t) \land c_1(t) = 0 \land c_2(t) = J_1(t-1)) \oplus \\
(F_1.c_2.0(t) \land c_1(t) = J_1(t-1) \land c_2(t) = 0)
\]

*Time dependent* Boolean fault variables

(Time dependent fault variables $F_1.c_1.0(t)$ and $F_1.c_2.0(t)$ in the concession ‘switch on’ the relevant fault.)

All very similar to what has gone before ...
Retrenchment Directed Analysis of a timed TLE

TLE is an expression in the external variables of the system.

Given a TLE, we can use the structure of the system to guide the decomposition and instantiation of the simulation relation. Assuming the TLE indeed represents a fault, we decompose the concession to derive a resolution tree (RT) for the TLE.

Eg. Timed TLE: \( J_1 = J_2 = P_1(t) = 1 \) (this means that: \( P_1 \) is constrained to be 1 at time \( t \) and is unconstrained at other times; \( J_1 \) and \( J_2 \) are constrained to be 1 at all times; and \( P_2 \) is regarded as irrelevant)

Search backwards from the outputs, as before.
The TLE.
Analyze $P_1(t) = 1$.

Two causes.

\[
C_{K0} \equiv O_{K2} \circ C_{K1} \lor C_{K2} \circ O_{K1} \lor C_{K2} \circ C_{K1}
\]

\[
C_{K1} \equiv O_{A2} \mid C_{A3} \lor C_{A2} \mid O_{A3} \lor C_{A2} \mid C_{A3}
\]
Left hand cause.
Trace through $F_3$, and through $A_1$.
Two causes.

$$C_{K2} \equiv O_{F3} \lor' C_{K3} \lor C_{F3} \lor' O_{K3} \lor C_{F3} \lor' C_{K3}$$

$$'C_{K3} \equiv 'O_{A1} \lor' C_{K4} \lor 'C_{A1} \lor' O_{K4} \lor 'C_{A1} \lor' C_{K4}$$

Pre-primes denote decrements in time relative to $t$. 
Left hand cause. Requires \( F1.c1.0 \) and \( 'F1.c2.0 \) (different times). Satisfiable.

\[
'c2 = 0 \land 'c3 = 1
\]

\[
'c2 = 1 \land 'c3 = 0
\]

\[
'c1 = 0 \land 'c6 = 1
\]

\[
'c1 = 1 \land 'c6 = 0
\]

\[
C_{K4} \equiv 'O_{F1} 'C_{F2} \lor 'C_{F1} 'O_{F2} \lor 'C_{F1} 'C_{F2}
\]

Pre-primes denote decrements in time relative to \( t \).
Right hand cause. Satisfiable. A basic cause of the TLE.

\[ C_{K4} \equiv O_{F1} | C_{F2} \lor C_{F1} | O_{F2} \lor C_{F1} | C_{F2} \]

Pre-primes denote decrements in time relative to \( t \).
... eventually ...
The full RT of the example TLE (compressed).
Minimisation

... done as before.
\begin{equation}
\begin{aligned}
c_1 &= 0 \land c_6 = 1 \\
c_1 &= 1 \land c_6 = 0
\end{aligned}
\end{equation}
Generating a Fault Tree

The resolution tree can be used to derive a FT according to standard FT conventions.

• Basic Fault nodes.
• Intermediate events.

Doing this on the timed example yields:

TLE: \((\exists P2 \cdot P1(t) = 1 \land J1 = 1 \land J2 = 1 \ldots)\)

c1,c5,c6 = 0,1,1

\(\neg c1 = 1\)
\(\neg c2 = 0\)
\(F1.c1\)
\(F1.c2\)

c1,c5,c6 = 0,1,1

\(\neg c1 = 1\)
\(\neg c3 = 0\)
\(F1.c1\)
\(F2.c3\)

c1,c6 = 1,0

\(c3 = 0\)
\(F3.c6\)

\(c1,c5,c6 = 1,0,0\)

\(\neg c2 = 0\)
\(\neg c3 = 0\)
\(F1.c2\)
\(F2.c3\)

\(c1 = 1\)
\(c2 = 0\)
Durable TLEs, Initial States, Cold Start Failures

Some questions:

What happens if the TLE mentions more than one output time value?

What happens if the states generated during the analysis are initial states?

Such questions can be addressed via a region-sensitive analysis.
Durable TLEs, Initial States, Cold Start Failures

Some questions:

What happens if the TLE mentions more than one output time value?

What happens if the states generated during the analysis are initial states?

Such questions can be addressed via a region-sensitive analysis.

A region is a connected set of nodes in the resolution analysis, all of which refer to the same time value.

Regions referring to the same time value, combined using AND/OR in the usual way, define what the analysis knows about specific states.
TLEs that mention more than one output time can be dealt with two ways:

1. Do a separate analysis for each needed time value and reconcile info about derived states later. Eg. exclude pair indicated by # conflict link opposite.

2. Modify description of the relevant retrenchments by parallel composing the copies needed for each relevant time value. In effect, precomputes the possibilities in 1.

To deal with initial states, can do the above for $t = 0 \ldots$ max_delay_in_circuit. Truncate any state labelled with $t = 0$, up to nearest ancestor disjunction, if it is inconsistent with initial state definition.

When we allow cyclic circuits and feedback, some issues/questions arise.

Retrenchment: What does retrenchment \textit{mean} in a feedback context?

Algorithmically: What do you do when a (top-down) backwards analysis of a looping circuit goes on forever?

Theoretically: What happens to a (bottom up) inductive process when there is no bottom (i.e. no base cases)?

When we allow cyclic circuits and feedback, some issues/questions arise.

Retrenchment: What does retrenchment mean in a feedback context? Need to define a suitable notion of retrenchment for backwards analysis of a looping circuit.

Algorithmically: What do you do when a (top-down) backwards analysis of a looping circuit goes on forever? Need to prevent the generation of infinite FTs.

Theoretically: What happens to a (bottom up) inductive process when there is no bottom (i.e. no base cases)? Need to replace standard induction by something else.
Retrenchment

Retrenchment for big acyclic systems is approached by composing retrenchments for their components.

Retrenchment for cyclic systems can be approached by breaking back-links, and composing arbitrary collections of (the causal relations for) the acyclic system that results. For example ...

Just composing a few copies of the acyclic causal relation is not enough, because arbitrarily many copies might be needed in any practical case.

Need something more comprehensive.
Retrenchment

You consider the input, output, and internal streams of the cyclic circuit.

You demand that the corresponding acyclic causal relation holds (for these streams) at all times that are reachable via a finite number of traversals (into the past) of the cyclic circuit. The collection of such constraints combines to form a causal relation for the cyclic circuit. ‘λ-constraints’ become global consistency constraints.

Likewise, you assemble within, output, concedes relations for a retrenchment between two such infinitary causal relations:

• The within relation assets the acyclic within relation at all appropriate past times.
• The output relation assets the acyclic output relation at all appropriate past times.
• The concedes relation assets that for every path into the past, the acyclic concedes relation holds at least once.

With such retrenchment data, an infinitary retrenchment PO can be proved sound (uses König’s Lemma).

Thus the appropriateness of retrenchment simulation extends to the infinite case.
Algorithmic Considerations

Since the resolution tree algorithm starts at the TLE, and works backwards, it can get started and run, regardless of whether the circuit is cyclic or not.

With a cyclic circuit, a non-terminating sequence of traversals of various loops in the circuit, naively generates an infinite fault tree.

An infinite fault tree is no use to engineers — engineers cannot manipulate or process arbitrary infinite objects.
Algorithmic Considerations

Since the resolution tree algorithm starts at the TLE, and works backwards, it can start to do so, regardless of whether the circuit is cyclic or not.

With a cyclic circuit, a non-terminating sequence of traversals of various loops in the circuit, naively generates an infinite fault tree.

An infinite fault tree is no use to engineers — engineers cannot manipulate or process arbitrary infinite objects.

However, if the fault tree has a periodic structure, we can represent it finitely using back-links ... this is about the best we might do.

To generate a periodic fault tree, the TLE must:
• constrain only finitely many output time values;
• have input constraints that (into the past) become eventually periodic.
Theoretical Considerations

Without feedback, the analysis process is guaranteed to terminate, whether viewed top-down or bottom-up.

The usual bottom-up inductive approach takes a collection of smaller instances (starting from base cases), and from their properties, infers the properties of a larger instance formed by composition of the smaller instances.

With feedback, a backwards analysis is not guaranteed to terminate, and in particular, there are no base cases to get an inductive treatment started.

An alternative to induction, for processes without base cases, is co-induction. Co-induction takes a large instance, and from its properties, infers the properties of a collection of smaller instances formed by decomposition of the larger instance (i.e. the opposite way round to induction).

However, the picture that emerges is so close to the algorithmic approach that a distinct treatment is not really cost effective.
10. FT Extraction for Feedback Circuits.

Modify the earlier example to introduce feedback aspects:

Behaviour ... (suppressing λ’s) ... of the components is exactly as before.
10. FT Extraction for Feedback Circuits.

Component retrenchments as before:

Adders (instantaneous, no failures):

\[ W_{A1}(\langle c_2(t), c_3(t) \rangle) \equiv \text{true} \]
\[ O_{A1}(c_5(t), \langle c_2(t), c_3(t) \rangle) \equiv (c_5(t) = c_2(t) + c_3(t)) \]
\[ C_{A1}(c_5(t), \langle c_2(t), c_3(t) \rangle) \equiv \text{false} \]

Fanouts (delay-1, at most one \textit{stuck\_at\_zero} fault active at any time):

\[ W_{F1}(J_1(t - 1)) \equiv \text{true} \]
\[ O_{F1}(\langle c_1(t), c_2(t) \rangle, J_1(t - 1)) \equiv (c_1(t) = c_2(t) = J_1(t - 1)) \]
\[ C_{F1}(\langle c_1(t), c_2(t) \rangle, J_1(t - 1)) \equiv (F1.c1.0(t) \land c_1(t) = 0 \land c_2(t) = J_1(t - 1)) \oplus (F1.c2.0(t) \land c_1(t) = J_1(t - 1) \land c_2(t) = 0) \]
10. FT Extraction for Feedback Circuits.

Eg. Timed cyclic TLE: $J_1 = P_1(t) = 1$ (this means that: $P_1$ is constrained to be 1 at time $t$ and is unconstrained at other times; $J_1$ is constrained to be 1 at all times)

Proceed as before ... as if the circuit was really (a suitably constrained version of):
The TLE.

$P_1(t) = 1 \land J_1 = 1$
Analyse $P1(t) = 1$.
Two causes.
Right hand cause.

Pre-primes denote decrements in time relative to $t$. 

$$J_1 = 1, P_1(t) = 1$$

$$c_6 = 0, \; \; \; c_5 = 0$$

$$c_1, c_6 = 0, 1$$

$$c_1, c_6 = 1, 0$$

$$c_1, c_6 = 0, 1$$

$$c_6 = 0$$

$$c_5 = 0$$
Equate $c_5 = 0$.

Pre-primes denote decrements in time relative to $t$. 

Analyse $c_5 = 0$. 

R. Banach, School of Computer Science, University of Manchester, UK
M. Bozzano, Fondazione Bruno Kessler, FBK-IRST, Tento, Italy
Analyse $\gamma e = 0$.

Pre-primes denote decrements in time relative to $t$. 
Analyze \( c5 = 0 \).
Dataflow examination shows that this is same situation as \( c5 = 0 \) node above, shifted in time by 1. Loop back.

Pre-primes denote decrements in time relative to \( t \).
... eventually ...
Full infinite FT, but with finite representation.

Input eventually periodic into the past guarantees regular structure of FT.
Back-Links, Durable TLEs, Initial States, Cold Start Failures

Input eventually periodic into the past guarantees eventual looping in FT.

Loops can be broken by installing back-links at will. Eg. ’c2, ye = 0,1 back-link; could be replaced by a ’c5 = 1 back-link, reducing size of tree.

Of course, not all backwards paths need be infinite ... they could terminate.

If the TLE mentions more than one output time value ... dealt with as before.

Cold start fails ... dealt with as before ... can check whether states generated during backwards analysis could be initial states.
11. FSAP and the Model Checking Approach to FT Extraction.

An overview of the algorithms for fault tree generation available in FSAP.

Algorithms based on model checking techniques.

- In this tutorial: focus on BDD-based routines.
- SAT-based routines exist as well.
Model Checking

Automated technique to verify a formal system model against a formal specification.

- Systems typically modeled as state transition systems.
- Specifications provided as temporal logic formulae.

Model checking provides a formal guarantee that a specification is obeyed.

(A counterexample trace is produced if the specification does not hold)

- Exhaustive technique compared to testing and simulation.

Major breakthrough with the introduction of symbolic model checking:

- Idea: manipulate sets of states and transitions.
- Efficient symbolic representations for the characteristic functions of such sets.

In the rest of this chapter: model checking techniques applied to FT generation.
**Binary Diagrams**

BDD = Binary Decision Diagram.
OBDD = Ordered BDD.
   (Built with a specific variable order)
ROBDD = Reduced OBDD.
   (Canonical form: elimination of redundancies)

(RO)BDDs are an efficient and compact representation for Boolean formulas.

Size of the BDD depend on variable order.

Set-theoretic operations as logical operators.

A BDD for the formula
\((a_1 \leftrightarrow a_2) \land (b_1 \leftrightarrow b_2)\).

Dashed = false, solid = true
BDD-based Algorithms for FTA

Different algorithms available:
- **Forward** (FWD).
- **Backward** (BWD).

Optimizations:
- **Dynamic cone of influence** (DCOI).
- **Dynamic pruning** (PRUN).
Cut Sets

Top Level Event

State variables

Failure Mode variables

History variables

History variables remember past failure events
\( (O_i \text{ is true if and only if } F_i \text{ is true at some point in the past}) \)

\[
\begin{align*}
O_i & \rightarrow \text{next}(O_i) \\
\neg O_i & \rightarrow (\text{next}(O_i) \leftrightarrow \text{next}(F_i))
\end{align*}
\]

Dual concept in the future: prophecy variables

F1 \land F2
CUT SET

Permanent fault
Sporadic fault
No fault
Once F1
Once F2
Once F3
Forward Algorithm

function FTA-Forward \((M, Tle)\):
1. \(M := \text{Extend}(M, R^o)\);
2. \(Reach := \mathcal{I} \cap (\varnothing = f)\);
3. \(Front := \mathcal{I} \cap (\varnothing = f)\);
4. \(\text{while} \ (Front \neq \emptyset) \ \text{do}\)
5. \(\ \text{temp} := \text{Reach};\)
6. \(\ \text{Reach} := \text{Reach} \cup \text{fwd\_img}(M, Front);\)
7. \(\ \text{Front} := \text{Reach} \setminus \text{temp};\)
8. \(\text{end while};\)
9. \(CS := \text{Proj}(\varnothing, \text{Reach} \cap Tle);\)
10. \(MCS := \text{Minimize}(CS);\)
11. \(\text{return } \text{Map}_{2 \rightarrow f}(MCS);\)
Forward Algorithm

function FTA-Forward \((\mathcal{M}, Tle)\)
1 \(\mathcal{M} := \text{Extend}(\mathcal{M}, \mathcal{R}^o)\);
2 \(\text{Reach} := \mathcal{I} \cap (q = f)\);
3 \(\text{Front} := \mathcal{I} \cap (q = f)\);
4 while (\text{Front} \neq \emptyset) do
5 \hspace{1em} \text{temp} := \text{Reach};
6 \hspace{1em} \text{Reach} := \text{Reach} \cup \text{fwd_img}(\mathcal{M}, \text{Front})
7 \hspace{1em} \text{Front} := \text{Reach} \setminus \text{temp};
8 end while;
9 \(CS := \text{Proj}(q, \text{Reach} \cap Tle)\);
10 \(MCS := \text{Minimize}(CS)\);
11 return \(\text{Map}_{2 \rightarrow \mathcal{L}}(MCS)\);
Forward Algorithm

```plaintext
function FTA-Forward (M, Tle)
M := Extend(M, R°);
Reach := I ∩ (q = \_);
Front := I ∩ (q = f);
while (Front ≠ \{\}) do
    temp := Reach;
    Reach := Reach ∪ \fwd_img(M, Front);
    Front := Reach \ temp;
end while;
CS := Proj(\_Reach ∩ Tle);
MCS := Minimize(CS);
return Map_{2→L}(MCS);
```
Forward Algorithm

function FTA-Forward (\(\mathcal{M}, Tle\))
\begin{align*}
1 & \quad \mathcal{M} := \text{Extend}(\mathcal{M}, \mathcal{R}^o); \\
2 & \quad \text{Reach} := \mathcal{I} \cap (a = f); \\
3 & \quad \text{Front} := \mathcal{I} \cap (a = f); \\
4 & \quad \text{while } (\text{Front} \neq \emptyset) \text{ do} \\
5 & \quad \quad \text{temp} := \text{Reach}; \\
6 & \quad \quad \text{Reach} := \text{Reach} \cup \text{fwd-img}(\mathcal{M}, \text{Front}); \\
7 & \quad \quad \text{Front} := \text{Reach}\setminus\text{temp}; \\
8 & \quad \text{end while}; \\
9 & \quad \text{CS} := \text{Proj}(a, \text{Reach} \cap Tle); \\
10 & \quad \text{MCS} := \text{Minimize}(\text{CS}); \\
11 & \quad \text{return } \text{Map}_{2 \rightarrow f}(\text{MCS});
\end{align*}
Forward Algorithm

function FTA-Forward($M, Tle$)
1. $M := \text{Extend}(M, \mathcal{R}^{\circ})$;
2. $\text{Reach} := I \cap (q = f)$;
3. $\text{Front} := I \cap (q = f)$;
4. while ($\text{Front} \neq \emptyset$) do
   temp := Reach;
   $\text{Reach} := \text{Reach} \cup f\text{wd}_\text{img}(M, \text{Front})$;
   $\text{Front} := \text{Reach} \setminus \text{temp}$;
end while;
5. $CS := \text{Proj}(q, \text{Reach} \cap Tle)$;
6. $MCS := \text{Minimize}(CS)$;
7. return $\text{Map}_{2 \rightarrow \mathcal{L}}(MCS)$;

Init
Forward Algorithm

function FTA-Forward ($M, Tle$)
1. $M := \text{Extend}(M, R^o)$;
2. $Reach := I \cap (q = f)$;
3. $Front := I \cap (q = f)$;
4. while ($Front \neq \emptyset$) do
5. \hspace{0.5cm} $temp := Reach$;
6. \hspace{0.5cm} $Reach := Reach \cup fwd_{img}(M, Front)$;
7. \hspace{0.5cm} $Front := Reach \setminus temp$;
8. end while;
9. $CS := Proj(q, Reach \cap Tle)$;
10. $MCS := \text{Minimize}(CS)$;
11. return $Map_{2\rightarrow 1}(MCS)$;
**Forward Algorithm**

```plaintext
function FTA-Forward (M, Tle)
1    \( M := \text{Extend}(M, R^o); \)
2    \( \text{Reach} := I \cap (a = f); \)
3    \( \text{Front} := I \cap (a = f); \)
4    \( \textbf{while} (\text{Front} \neq \emptyset) \textbf{do} \)
5        \( \text{temp} := \text{Reach}; \)
6        \( \text{Reach} := \text{Reach} \cup \)
7            \( \text{fwd} \_ \text{img}(M, \text{Front}); \)
8        \( \text{Front} := \text{Reach}\setminus \text{temp}; \)
9    \( \textbf{end while}; \)
10   \( \text{CS} := \text{Proj}(a, \text{Reach} \cap \text{Tle}); \)
11   \( \text{MCS} := \text{Minimize}(\text{CS}); \)
12   \( \text{return } \text{Map}_2 \_ f(\text{MCS}); \)
```

R. Banach, School of Computer Science, University of Manchester, UK
M. Bozzano, Fondazione Bruno Kessler, FBK-IRST, Trento, Italy
Forward Algorithm

function FTA-Forward (M, Tle)
1   \( M := \text{Extend}(M, R^o); \)
2   \( \text{Reach} := I \cap (q = f); \)
3   \( \text{Front} := I \cap (q = f); \)
4   \( \text{while} (\text{Front} \neq \emptyset) \) do
5       \( \text{temp} := \text{Reach}; \)
6       \( \text{Reach} := \text{Reach} \cup \)
7               \( \text{fwd_img}(M, \text{Front}); \)
8       \( \text{Front} := \text{Reach} \setminus \text{temp}; \)
9   \( \text{end while}; \)
10  \( CS := \text{Proj}(\underline{a}, \text{Reach} \cap \text{Tle}); \)
11  \( \text{MCS} := \text{Minimize}(CS); \)
12  \( \text{return} \ Map_{\underline{a} \rightarrow \xi}(\text{MCS}); \)
function FTA-Forward ($M, Tle$)
1 $M := Ext(M, R^0)$;
2 $\text{Reach} := \mathcal{I} \cap (q = f)$;
3 $\text{Front} := \mathcal{I} \cap (q = f)$;
4 while ($\text{Front} \neq \emptyset$) do
5 $\text{temp} := \text{Reach}$;
6 $\text{Reach} := \text{Reach} \cup f_{\text{wd}_{\text{img}}}(M, \text{Front})$;
7 $\text{Front} := \text{Reach} \setminus \text{temp}$;
8 end while;
9 $CS := \text{Proj}(\mathcal{I}, \text{Reach} \cap Tle)$;
10 $MCS := \text{Minimize}(CS)$;
11 return $\text{Map}_{2 \rightarrow f}(MCS)$;
function FTA-Forward (M, Tle)
1   M := Extend(M, R°);
2   Reach := I \cap (a = f);
3   Front := I \cap (a = f);
4   while (Front ≠ ∅) do
5       temp := Reach;
6       Reach := Reach ∪ fwd_img(M, Front);
7       Front := Reach \temp;
8   end while;
9   CS := Proj(a, Reach \ Tle);
10  MCS := Minimize(CS);
11  return Map_{2→f}(MCS);
function FTA-Forward ($\mathcal{M}, Tle$)
1. $\mathcal{M} := \text{Extend}(\mathcal{M}, \mathcal{R}^o)$;
2. $\text{Reach} := \mathcal{I} \cap (\alpha = f)$;
3. $\text{Front} := \mathcal{I} \cap (\alpha = f)$;
4. while ($\text{Front} \neq \emptyset$) do
   5. $\text{temp} := \text{Reach}$;
   6. $\text{Reach} := \text{Reach} \cup \text{fwd_img}(\mathcal{M}, \text{Front})$;
   7. $\text{Front} := \text{Reach}\setminus\text{temp}$;
5. end while;
6. $CS := \text{Proj}(\alpha, \text{Reach} \cap Tle)$;
7. $MCS := \text{Minimize}(CS)$;
8. return $\text{Map}_{\alpha \rightarrow \mathcal{L}}(MCS)$;
function FTA-Forward (M, Tle)
1   $M := \text{Extend}(M, \mathcal{R}^\circ)$;
2   $\text{Reach} := \mathcal{I} \cap (\mathcal{Q} = f)$;
3   $\text{Front} := \mathcal{I} \cap (\mathcal{Q} = \overline{f})$;
4   while ($\text{Front} \neq \emptyset$) do
5       $\text{temp} := \text{Reach}$;
6       $\text{Reach} := \text{Reach} \cup \text{fwd} \text{-img}(M, \text{Front})$;
7       $\text{Front} := \text{Reach} \setminus \text{temp}$;
8   end while;
9   $CS := \text{Proj}(\mathcal{Q}, \text{Reach} \cap \text{Tle})$;
10  $MCS := \text{Minimize}(CS)$;
11  return $\text{Map}_{\mathcal{Q} \rightarrow \mathcal{L}}(MCS)$;
function FTA-Forward ($\mathcal{M}, Tle$)
1 \hspace{5mm} \mathcal{M} := \text{Extend}(\mathcal{M}, \mathcal{R}^\circ); \\
2 \hspace{5mm} \mathcal{R} := \mathcal{I} \cap (\underline{\mathcal{L}} = \underline{f}); \\
3 \hspace{5mm} \text{Front} := \mathcal{I} \cap (\underline{\mathcal{L}} = \underline{f}); \\
4 \hspace{5mm} \textbf{while} (\text{Front} \neq \emptyset) \hspace{5mm} \textbf{do} \\
5 \hspace{10mm} \text{temp} := \text{Reach}; \\
6 \hspace{10mm} \text{Reach} := \text{Reach} \cup \text{fwdimg}(\mathcal{M}, \text{Front}); \\
7 \hspace{10mm} \text{Front} := \text{Reach} \setminus \text{temp}; \\
8 \hspace{5mm} \textbf{end while}; \\
9 \hspace{5mm} \mathcal{C} := \text{Proj}(\underline{L}, \text{Reach} \cap Tle); \\
10 \hspace{5mm} \mathcal{MCS} := \text{Minimize}(\mathcal{C}); \\
11 \hspace{5mm} \text{return Map}_{2 \rightarrow \mathcal{L}}(\mathcal{MCS});

<table>
<thead>
<tr>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>O1</th>
<th>O2</th>
<th>O3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
**Forward Algorithm**

<table>
<thead>
<tr>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**function** FTA-Forward \( (\mathcal{M}, Tle) \)

1. \( \mathcal{M} := \text{Extend}(\mathcal{M}, \mathcal{R}^o); \)
2. \( \text{Reach} := \mathcal{I} \cap (a = f); \)
3. \( \text{Front} := \mathcal{I} \cap (a = f); \)
4. while \((\text{Front} \neq \emptyset)\) do
5. \( \text{temp} := \text{Reach}; \)
6. \( \text{Reach} := \text{Reach} \cup \text{fwd}\_\text{img}(\mathcal{M}, \text{Front}); \)
7. \( \text{Front} := \text{Reach} \setminus \text{temp}; \)
8. end while;
9. \( \text{CS} := \text{Proj}(a, \text{Reach} \cap \text{Tle}); \)
10. \( \text{MCS} := \text{Minimize}(\text{CS}); \)
11. return \( \text{Map}_{2\rightarrow \mathcal{L}}(\text{MCS}); \)

R. Banach, School of Computer Science, University of Manchester, UK
M. Bozzano, Fondazione Bruno Kessler, FBK-IRST, Trento, Italy
function FTA-Forward (M, Tle)
1 \( M := \text{Extend}(M, R^\circ) \);
2 \( \text{Reach} := \mathcal{I} \cap (q = f) \);
3 \( \text{Front} := \mathcal{I} \cap (q = f) \);
4 while \( \text{Front} \neq \emptyset \) do
5 \hspace{1em} \text{temp} := \text{Reach};
6 \hspace{1em} \text{Reach} := \text{Reach} \cup \text{fwd_img}(M, \text{Front});
7 \hspace{1em} \text{Front} := \text{Reach} \setminus \text{temp};
8 \hspace{1em} end while;
9 \hspace{1em} \text{CS} := \text{Proj}(q, \text{Reach} \cap \text{Tle});
10 \hspace{1em} \text{MCS} := \text{Minimize} \text{(CS)};
11 \hspace{1em} return Map_{2 \rightarrow \mathcal{L}} \text{(MCS)};
Forward Algorithm

function FTA-Forward \( (\mathcal{M}, Tle) \)
1 \( \mathcal{M} := \text{Extend}(\mathcal{M}, \mathcal{R}^o); \)
2 \( \text{Reach} := \mathcal{I} \cap (a = f); \)
3 \( \text{Front} := \mathcal{I} \cap (a = f); \)
4 while \( (\text{Front} \neq \emptyset) \) do
5 \( \text{temp} := \text{Reach}; \)
6 \( \text{Reach} := \text{Reach} \cup \text{fwd}_{\text{img}}(\mathcal{M}, \text{Front}); \)
7 \( \text{Front} := \text{Reach} \setminus \text{temp}; \)
8 end while;
9 \( CS := \text{Proj}(a, \text{Reach} \cap Tle); \)
10 \( \text{MCS} := \text{Minimize}(CS); \)
11 return \( \text{Map}_{a \rightarrow f}(\text{MCS}); \)

R. Banach, School of Computer Science, University of Manchester, UK
M. Bozzano, Fondazione Bruno Kessler, FBK-IRST, Trento, Italy
Forward Algorithm

```
function FTA-Forward (M, Tle)
1  \( M := \text{Extend}(M, R^o); \)
2  \( \text{Reach} := I \cap (q = f); \)
3  \( \text{Front} := I \cap (q = f); \)
4  \while (Front \neq \emptyset) \do \\
5      \text{temp} := \text{Reach}; \\
6      \text{Reach} := \text{Reach} \cup \\
7        \text{fwd_img}(M, \text{Front}); \\
8      \text{Front} := \text{Reach} \setminus \text{temp}; \\
9  \end \while; \\
10 \text{CS} := \text{Proj}(q, \text{Reach} \cap Tle); \\
11 \text{MCS} := \text{Minimize}(\text{CS}); \\
12 \text{return Map}_{2\to \underline{c}}(\text{MCS}); 
```
function FTA-Forward $(M, \text{Tle})$

1. $M := \text{Extend}(M, \mathcal{R}^o)$;
2. $\text{Reach} := \mathcal{I} \cap (\mathcal{g} = f)$;
3. $\text{Front} := \mathcal{I} \cap (\mathcal{g} = f)$;
4. while ($\text{Front} \neq \emptyset$) do
   1. $\text{temp} := \text{Reach};$
   2. $\text{Reach} := \text{Reach} \cup \text{fwd_img}(M, \text{Front});$
   3. $\text{Front} := \text{Reach}\setminus\text{temp};$
5. end while;
6. $CS := \text{Proj}(\mathcal{g}, \text{Reach} \cap \text{Tle});$
7. $MCS := \text{Minimize}(CS);$
8. return $\text{Map}_{\mathcal{g} \rightarrow f}(MCS);$
function FTA-Forward (M, Tle)  
1 \( M := \text{Extend}(M, R^o); \)  
2 \( \text{Reach} := \mathcal{I} \cap (q = f); \)  
3 \( \text{Front} := \mathcal{I} \cap (q = f); \)  
4 \( \text{while } (\text{Front} \neq \emptyset) \text{ do} \)  
5 \( \text{temp} := \text{Reach}; \)  
6 \( \text{Reach} := \text{Reach} \cup \text{fwdimg}(M, \text{Front}); \)  
7 \( \text{Front} := \text{Reach} \setminus \text{temp}; \)  
8 \( \text{end while}; \)  
9 \( \text{CS} := \text{Proj}(q, \text{Reach} \cap Tle); \)  
10 \( \text{MCS} := \text{Minimize}(``CS); \)  
11 \( \text{return } \text{Map}_{\phi \rightarrow f}(\text{MCS}); \)
Backward Algorithm

function FTA-Backward (M, Tle)
1  M := Extend(M, R^g);
2  Reach := Tle \cap (g = \bar{f});
3  Front := Tle \cap (g = f);
4  while (Front \neq \emptyset) do
5    temp := Reach;
6    Reach := Reach \cup 
              bwd_img(M, Front);
7    Front := Reach \setminus temp;
8  end while;
9  CS := Proj(g, Reach \cap \mathcal{I});
10  MCS := Minimize(CS);
11  return Map_{g \rightarrow \bar{f}}(MCS);
Backward Algorithm

function FTA-Backward (M, Tle)
1. \( M := \text{Extend}(M, R^g); \)
2. \( \text{Reach} := \text{Tle} \cap (g = f); \)
3. \( \text{Front} := \text{Tle} \cap (g = f); \)
4. \( \text{while } (\text{Front} \neq \emptyset) \text{ do} \)
5. \( \text{temp} := \text{Reach}; \)
6. \( \text{Reach} := \text{Reach} \cup \text{bwd_img}(M, \text{Front}); \)
7. \( \text{Front} := \text{Reach}\setminus\text{temp}; \)
8. \( \text{end while}; \)
9. \( \text{CS} := \text{Proj}(g, \text{Reach} \cap I); \)
10. \( \text{MCS} := \text{Minimize}(\text{CS}); \)
11. \( \text{return } \text{Map}_{g \rightarrow f}(\text{MCS}); \)
Backward Algorithm

function FTA-Backward (M, Tle)
1 \[ M := \text{Extend}(M, R^g); \]
2 \[ \text{Reach} := Tle \cap (g = f); \]
3 \[ \text{Front} := Tle \cap (g = f); \]
4 \[ \text{while (Front} \neq \emptyset) \text{ do} \]
5 \[ \text{temp} := \text{Reach}; \]
6 \[ \text{Reach} := \text{Reach} \cup \text{bwdImg}(M, \text{Front}); \]
7 \[ \text{Front} := \text{Reach}\backslash\text{temp}; \]
8 \[ \text{end while}; \]
9 \[ CS := \text{Proj}(g, \text{Reach} \cap I); \]
10 \[ MCS := \text{Minimize}(CS); \]
11 \[ \text{return } \text{Map}_{g\rightarrow f}(MCS); \]
function FTA-Backward \( (\mathcal{M}, Tle) \)
1 \( \mathcal{M} := \text{Extend}(\mathcal{M}, R^g) \);
2 \( \text{Reach} := Tle \cap (g = f) \);
3 \( \text{Front} := Tle \cap (g = f) \);
4 \textbf{while} (\text{Front} \neq \emptyset) \textbf{do}
5 \quad \text{temp} := \text{Reach} ;
6 \quad \text{Reach} := \text{Reach} \cup \text{bwdimg}(\mathcal{M}, \text{Front}) ;
7 \quad \text{Front} := \text{Reach} \setminus \text{temp} ;
8 \textbf{end while} ;
9 \quad \text{CS} := \text{Proj}(g, \text{Reach} \cap \mathcal{T}) ;
10 \quad \text{MCS} := \text{Minimize} \left( \text{CS} \right) ;
11 \quad \text{return} \ Map_{g \rightarrow f}(\text{MCS}) ;
Backward Algorithm

function FTA-Backward \((\mathcal{M}, Tle)\)
\begin{align*}
1 & \quad \mathcal{M} := \text{Extend}(\mathcal{M}, \mathcal{R}^g); \\
2 & \quad \text{Reach} := Tle \cap (g = f); \\
3 & \quad \text{Front} := Tle \cap (g = f); \\
4 & \textbf{while} (\text{Front} \neq \emptyset) \textbf{do} \\
5 & \quad \text{temp} := \text{Reach}; \\
6 & \quad \text{Reach} := \text{Reach} \cup \\
7 & \quad \quad \quad \\ & \quad \quad \quad \quad \text{bd}_{\mathcal{M}}(\mathcal{M}, \text{Front}); \\
8 & \quad \text{Front} := \text{Reach} \setminus \text{temp}; \\
9 & \textbf{end while}; \\
10 & \quad \text{CS} := \text{Proj}(g, \text{Reach} \cap I); \\
11 & \quad \text{MCS} := \text{Minimize}(\text{CS}); \\
12 & \quad \text{return} \text{Map}_{g \rightarrow f}(\text{MCS});
\end{align*}
function FTA-Backward ($\mathcal{M}, Tle$)
1 $\mathcal{M} := \text{Extend}(\mathcal{M}, \mathcal{R}^g)$;
2 $\text{Reach} := Tle \cap (g = f)$;
3 $\text{Front} := Tle \cap (g = f)$;
4 while ($\text{Front} \neq \emptyset$) do
5 $\text{temp} := \text{Reach}$;
6 $\text{Reach} := \text{Reach} \cup$
7 $\quad \text{bwdimg}(\mathcal{M}, \text{Front})$;
8 $\text{Front} := \text{Reach} \setminus \text{temp}$;
9 end while;
10 $CS := \text{Proj}(g, \text{Reach} \cap \mathcal{I})$;
11 $MCS := \text{Minimize}(CS)$;
12 return $\text{Map}_{g \rightarrow f}(MCS)$;
function FTA-Backward \((\mathcal{M}, \text{Tle})\)

1. \(\mathcal{M} := \text{Extend}(\mathcal{M}, \mathcal{R}^g)\);
2. \(\text{Reach} := \text{Tle} \cap (g = f)\);
3. \(\text{Front} := \text{Tle} \cap (g = f)\);
4. while \((\text{Front} \neq \emptyset)\) do
5. \(\text{temp} := \text{Reach}\);
6. \(\text{Reach} := \text{Reach} \cup \text{bwd}_{\text{img}}(\mathcal{M}, \text{Front})\);
7. \(\text{Front} := \text{Reach}\setminus\text{temp}\);
end while;
8. \(\text{CS} := \text{Proj}(g, \text{Reach} \cap \mathcal{T})\);
9. \(\text{MCS} := \text{Minimize}(\text{CS})\);
10. return \(\text{Map}_{g \rightarrow f}(\text{MCS})\);
Backward Algorithm

function FTA-Backward ($M, Tle$)
1. $M := \text{Extend}(M, R^g)$;
2. $Reach := Tle \cap (g = f)$;
3. $Front := Tle \cap (g = f)$;
4. while ($Front \neq \emptyset$) do
5. \quad temp := Reach;
6. \quad Reach := Reach \cup \text{bwd$\_img$($M, Front$)};
7. \quad Front := Reach \setminus \text{temp};
8. end while;
9. $CS := \text{Proj}(g, Reach \cap T)$;
10. $MCS := \text{Minimize}(CS)$;
11. return $\text{Map}_{g \rightarrow f}(MCS)$;
Backward Algorithm

\begin{algorithm}
\textbf{function} FTA-Backward ($\mathcal{M}, Tle$) \\
1 \hspace{1em} $\mathcal{M} := \text{Extend}(\mathcal{M}, R^g)$; \\
2 \hspace{1em} $\text{Reach} := Tle \cap (g = f)$; \\
3 \hspace{1em} $\text{Front} := Tle \cap (g = f)$; \\
4 \hspace{1em} \textbf{while} ($\text{Front} \neq \emptyset$) \textbf{do} \\
5 \hspace{2em} $\text{temp} := \text{Reach}$; \\
6 \hspace{2em} $\text{Reach} := \text{Reach} \cup$ \\
7 \hspace{3em} $\text{bwd}_{\text{img}}(\mathcal{M}, \text{Front})$; \\
8 \hspace{2em} $\text{Front} := \text{Reach} \setminus \text{temp}$; \\
9 \hspace{1em} \textbf{end while}$; \\
10 \hspace{1em} $CS := \text{Proj}(g, \text{Reach} \cap \mathcal{I})$; \\
11 \hspace{1em} $MCS := \text{Minimize}(CS)$; \\
12 \hspace{1em} \textbf{return} $\text{Map}_{g-f}(MCS)$;
\end{algorithm}
Backward Algorithm

function FTA-Backward (\(\mathcal{M}, Tle\))
1. \(\mathcal{M} := \text{Extend}(\mathcal{M}, R^g)\);
2. \(\text{Reach} := Tle \cap (\overline{g} = \overline{f})\);
3. \(\text{Front} := Tle \cap (\overline{g} = \overline{f})\);
4. while (Front \(\neq \emptyset\)) do
5. \(\text{temp} := \text{Reach};\)
6. \(\text{Reach} := \text{Reach} \cup bwd_img(\mathcal{M}, \text{Front});\)
7. \(\text{Front} := \text{Reach} \setminus \text{temp};\)
8. end while;
9. \(\text{CS} := \text{Proj}(\overline{g}, \text{Reach} \cap I);\)
10. \(\text{MCS} := \text{Minimize}(\text{CS});\)
11. return \(\text{Map}_{\overline{g}}^{\overline{f}}(\text{MCS});\)
function FTA-Backward \((M, Tle)\)
1. \(M := \text{Extend}(M, R^g)\);
2. \(Reach := Tle \cap (g = f)\);
3. \(Front := Tle \cap (\overline{g} = f)\);
4. \textbf{while} \((Front \neq \emptyset)\) \textbf{do}
5. \hspace{1em} \(temp := Reach\);
6. \hspace{1em} \(Reach := Reach \cup \text{bwd_img}(M, Front)\);
7. \hspace{1em} \(Front := Reach \setminus temp\);
8. \hspace{1em} \textbf{end while};
9. \(CS := \text{Proj}(g \overline{Reach \cap I})\);
10. \(MCS := \text{Minimize}(CS)\);
11. \textbf{return} \(\text{Map}_{g \rightarrow f}(MCS)\);
Backward Algorithm

```
function FTA-Backward (M, Tle)
1   M := Extend(M, R^g);
2   Reach := Tle \cap (g = f);
3   Front := Tle \cap (g = f);
4   while (Front ≠ ∅) do
5       temp := Reach;
6       Reach := Reach ∪
7           bwd_img(M, Front);
8       Front := Reach \temp;
9   end while;
10   CS := Proj(g \ Reach \cap I);
11   MCS := Minimize(CS);
12   return Map_{g→f}(MCS);
```
Backward Algorithm

function FTA-Backward \((\mathcal{M}, Tle)\)
1. \(\mathcal{M} := \text{Extend}(\mathcal{M}, R^g)\);
2. \(\text{Reach} := Tle \cap \{q = f\}\);
3. \(\text{Front} := Tle \cap \{q = f\}\);
4. while \((\text{Front} \neq \emptyset)\) do
   5. \(\text{temp} := \text{Reach}\);
   6. \(\text{Reach} := \text{Reach} \cup \text{bwd_img}(\mathcal{M}, \text{Front})\);
   7. \(\text{Front} := \text{Reach} \setminus \text{temp}\);
end while;
8. \(\text{CS} := \text{Proj}(g, \text{Reach} \cap I)\);
9. \(\text{MCS} := \text{Minimize}(\text{CS})\);
10. return \(\text{Map}_{g \rightarrow f}(\text{MCS})\);

And so on ...
Dynamic Cone of Influence

\[ Tle \]

function $FTA$-Backward-DCOI ($\mathcal{M}, Tle$)
1 \hspace{1cm} $i := 0$;
2 \hspace{1cm} $\mathcal{M} := Extend(\mathcal{M}, R^g)$;
3 \hspace{1cm} $\text{Reach} := Tle \cap (g = f)$;
4 \hspace{1cm} $\text{Front} := Tle \cap (g = \overline{f})$;
5 \hspace{1cm} while ($\text{Front} \neq \emptyset$) do
6 \hspace{2cm} $\text{temp} := \text{Reach}$;
7 \hspace{2cm} $\mathcal{M}^i = \text{dcoi\_get}(\mathcal{M}, Tle, i)$;
8 \hspace{2cm} $\text{Reach} := \text{Reach} \cup$
9 \hspace{3cm} $\text{bwd\_img}(\mathcal{M}^i, \text{Front})$;
10 \hspace{2cm} $\text{Front} := \text{Reach} \setminus \text{temp}$;
11 \hspace{1cm} $i := i + 1$;
12 \hspace{1cm} end while;
13 \hspace{1cm} $CS := Proj(g, \text{Reach} \cap \mathcal{T})$;
14 \hspace{1cm} $MCS := \text{Minimize}(CS)$;
15 \hspace{1cm} return $Map_{g \rightarrow \overline{f}}(MCS)$;
Dynamic Cone of Influence

\[ Tle \]

Compute pre-images & restricted Kripke structures, based on dependency with \( Tle \)

\[ M^0 \leq M^1 \leq \ldots \leq M^{n-1} \leq M^n \]

- defer construction of the Kripke structure
- hopefully \( M^n \) is smaller than the global \( M \)

\begin{verbatim}
function FTA-Backward-DCOI (M, Tle)
    i := 0;
    \( M := \text{Extend}(M, R^g) \);
    Reach := Tle \cap \{ g = f \};
    Front := Tle \cap \{ g = \overline{f} \};
    while (Front \neq \emptyset) do
        temp := Reach;
        \( M' := \text{dcoi-get}(M, Tle, i) \);
        Reach := Reach \cup \text{bwd_img}(M', Front);
        Front := Reach \setminus temp;
        i := i + 1;
    end while;
    CS := \text{Proj}(g, \text{Reach} \cap \mathcal{T}) ;
    MCS := \text{Minimize}(CS) ;
    return Map_{g \rightarrow \mathcal{F}}(MCS) ;
\end{verbatim}
Dynamic Cone of Influence

Compute pre-images & restricted Kripke structures, based on dependency with Tle

\[ M^0 \leq M^1 \leq \ldots \leq M^{n-1} \leq M^n \]

- defer construction of the Kripke structure
- hopefully \( M^n \) is smaller than the global \( M \)

function FTA-Backward-DCOI (\( M, Tle \))
1 \hspace{1em} \( i := 0; \)
2 \hspace{1em} \( M := \text{Extend}(M, R^g); \)
3 \hspace{1em} \( \text{Reach} := \text{Tle} \cap (g = f); \)
4 \hspace{1em} \( \text{Front} := \text{Tle} \cap (g = \bar{f}); \)
5 \hspace{1em} while (Front ≠ \emptyset) do
6 \hspace{2em} \( \text{temp} := \text{Reach}; \)
7 \hspace{2em} \( M^i := \text{dcoi_get}(M, Tle, i); \)
8 \hspace{2em} \( \text{Reach} := \text{Reach} \cup \text{Build_img}(M^i, \text{Front}); \)
9 \hspace{2em} \( \text{Front} := \text{Reach} \setminus \text{temp}; \)
10 \hspace{2em} \( i := i + 1; \)
11 \hspace{1em} end while;
12 \hspace{1em} \( CS := \text{Proj}(g, \text{Reach} \cap I); \)
13 \hspace{1em} \( MCS := \text{Minimize}(CS); \)
14 \hspace{1em} return \( \text{Map}_w (\bar{M}(MCS); \)
Dynamic Cone of Influence

Compute pre-images & restricted Kripke structures, based on dependency with $Tle$

$M^0 \subseteq M^1 \subseteq \cdots \subseteq M^{n-1} \subseteq M^n$

- defer construction of the Kripke structure
- hopefully $M^n$ is smaller than the global $M$

```python
function FTA-Backward-DCOI ($M, Tle$)
    $i := 0$
    $M := \text{Extend}(M, R^g)$
    $Reach := Tle \cap (q = f)$
    $Front := Tle \cap (q = f)$
    while ($Front \neq \emptyset$) do
        $temp := Reach$
        $M^i := \text{dcoi_get}(M, Tle, i)$
        $ Reach := Reach \cup \text{build_img}(M^i, Front)$
        $ Front := Reach \setminus temp$
        $i := i + 1$
    end while
    $CS := \text{Proj}(q, Reach \cap \tau)$
    $MCS := \text{Minimize}(CS)$
    return $Map_{\exists \tau}(MCS)$
```
Dynamic Cone of Influence

Compute pre-images & restricted Kripke structures, based on dependency with \( Tle \)
\[ M^0 \preceq M^1 \preceq \ldots \preceq M^{n-1} \preceq M^n \]

- defer construction of the Kripke structure
- hopefully \( M^n \) is smaller than the global \( M \)

function FTA-Backward-DCOI (\( M, Tle \))
1. \( i := 0; \)
2. \( M := \text{Ext}(M, R^g); \)
3. \( \text{Reach} := Tle \cap (q = f); \)
4. \( \text{Front} := Tle \cap (g = f); \)
5. while (Front ≠ ∅) do
   6. \( \text{temp} := \text{Reach}; \)
   7. \( \text{M}^i := \text{dcoi-get}(M, Tle, i); \)
   8. \( \text{Reach} := \text{Reach} \cup \text{bdwimg}(\text{M}^i, \text{Front}); \)
   9. \( \text{Front} := \text{Reach} \setminus \text{temp}; \)
   10. \( i := i + 1; \)
6. end while;
11. \( \text{CS} := \text{Proj}(q, \text{Reach} \cap \mathcal{T}); \)
12. \( \text{MCS} := \text{Minimize}(\text{CS}); \)
13. return Map_{2-L}(\text{MCS});
Dynamic Cone of Influence

Compute pre-images & restricted Kripke structures, based on dependency with $T\ell e$

$M^0 \subseteq M^1 \subseteq \ldots \subseteq M^{n-1} \subseteq M^n$

- defer construction of the Kripke structure
- hopefully $M^n$ is smaller than the global $M$

**function** FTA-Backward-DCOI ($\mathcal{M}, T\ell e$)

1. $i := 0$;
2. $\mathcal{M} := \text{Extend}(\mathcal{M}, R^g)$;
3. $\text{Reach} := T\ell e \cap (g = f)$;
4. $\text{Front} := T\ell e \cap (g = \overline{f})$;
5. **while** ($\text{Front} \neq \emptyset$) **do**
   6. $\text{temp} := \text{Reach}$;
   7. $\boxed{\mathcal{M}^i := \text{dcoi-get}(\mathcal{M}, T\ell e, i)}$;
   8. $\text{Reach} := \text{Reach} \cup \text{build}(\mathcal{M}^i, \text{Front})$;
   9. $\text{Front} := \text{Reach} \setminus \text{temp}$;
   10. $i := i + 1$;
6. **end while**;
12. $\text{CS} := \text{Proj}(g, \text{Reach} \cap \mathcal{I})$;
13. $\text{MCS} := \text{Minimize}(\text{CS})$;
14. **return** $\text{Map}_{\Delta - \mathcal{L}}(\text{MCS})$;
Dynamic Pruning

```
function FTA-Forward-Pruning (\mathcal{M}, \text{Tle})
1    \text{CS} := \emptyset;
2    \mathcal{M} := \text{Extend}(\mathcal{M}, \mathcal{R}^o);
3    \text{Reach} := \mathcal{I} \cap (\alpha = f);
4    \text{Front} := \mathcal{I} \cap (\alpha = \bar{f});
5     \textbf{while} (\text{Front} \neq \emptyset) \textbf{do}
6     \quad \text{CS} := \text{CS} \cup \text{Proj}(\alpha, \text{Reach} \cap \text{Tle});
7     \quad \text{temp} := \text{Reach};
8     \quad \text{Reach} := \text{Reach} \cup \text{fwd-img}(\mathcal{M}, \text{Front});
9     \quad \text{Front} := \text{Reach} \setminus \text{temp};
10    \quad \text{Front} := \text{Front} \setminus \text{Widen} \text{(CS)};
11 \textbf{end while};
12    \text{MCS} := \text{Minimize}(\text{CS});
13 \textbf{return} Map_{\alpha \rightarrow f} (\text{MCS});
```

Dynamic Pruning

function FTA-Forward-Pruning (M, Tle)
1   CS := ∅;
2   M := Extend(M, R°);
3   Reach := I ∩ (q = f);
4   Front := I ∩ (q = f);
5   while (Front ≠ ∅) do
6       CS := CS ∪ Proj(q, Reach ∩ Tle);
7       temp := Reach;
8       Reach := Reach ∪ fwd_img(M, Front);
9       Front := Reach \ temp;
10      Front := Front \ Widen(CS);
11  end while;
12  MCS := Minimize(CS);
13  return Map_q→f(MCS);

At each iteration, compute a partial set of cut sets
Use the partial set to prune non-minimal config.
in the search space
Dynamic Pruning

function FTA-Forward-Pruning (\(\mathcal{M}, Tle\))
1. \(CS := \emptyset;\)
2. \(\mathcal{M} := \text{Extend}(\mathcal{M}, \mathcal{R}^\circ);\)
3. \(\text{Reach} := \mathcal{I} \cap (q = f);\)
4. \(\text{Front} := \mathcal{I} \cap (q = f);\)
5. \textbf{while} (\(\text{Front} \neq \emptyset\)) \textbf{do}
6. \(CS := CS \cup \text{Proj}(q, \text{Reach} \cap Tle);\)
7. \(\text{temp} := \text{Reach};\)
8. \(\text{Reach} := \text{Reach} \cup \text{fwd\_img}(\mathcal{M}, \text{Front});\)
9. \(\text{Front} := \text{Reach}\setminus\text{temp};\)
10. \(\text{Front} := \text{Front}\setminus\text{Widen}(CS);\)
11. \textbf{end while;}
12. \(MCS := \text{Minimize}(CS);\)
13. \textbf{return} \(\text{Map}_{q \rightarrow f}(MCS);\)

At each iteration, compute a partial set of cut sets
Use the partial set to prune non-minimal config.
in the search space

R. Banach, School of Computer Science, University of Manchester, UK
M. Bozzano, Fondazione Bruno Kessler, FBK-IRST, Trento, Italy
Dynamic Pruning

function FTA-Forward-Pruning \((M, Tle)\)
1. \(CS := \emptyset;\)
2. \(M := \text{Extend}(M, R^\circ);\)
3. \(\text{Reach} := \mathcal{I} \cap (a = f);\)
4. \(\text{Front} := \mathcal{I} \cap (a = f);\)
5. \(\text{while } (\text{Front} \neq \emptyset) \text{ do}\)
6. \(\text{CS} := \text{CS} \cup \text{Proj}(a, \text{Reach} \cap Tle);\)
7. \(\text{temp} := \text{Reach};\)
8. \(\text{Reach} := \text{Reach} \cup \text{fwd_img}(M, \text{Front});\)
9. \(\text{Front} := \text{Reach} \setminus \text{temp};\)
10. \(\text{Front} := \text{Front} \setminus \text{Widen}(\text{CS});\)
6. \(\text{end while};\)
12. \(MCS := \text{Minimize(}\text{CS});\)
13. \(\text{return } Map_{\overline{a} \rightarrow f}(MCS);\)

At each iteration, compute a partial set of cut sets
Use the partial set to prune non-minimal config.
in the search space
Dynamic Pruning

```
function FTA-Forward-Pruning (M, Tle) 
1. CS := ∅;
2. M := Extend(M, R°);
3. Reach := I ∩ (q = f);
4. Front := I ∩ (q = f);
5. while (Front ≠ ∅) do
6.   CS := CS ∪ Proj(q, Reach ∩ Tle);
7.   temp := Reach;
8.   Reach := Reach ∪
9.      fwd_img(M, Front);
10.  Front := Reach\temp;
11.  Front := Front\Widen(CS);
12. end while;
13. MCS := Minimize(CS);
14. return Map_q→f(MCS);
```

At each iteration, compute a partial set of cut sets
Use the partial set to prune non-minimal config.
in the search space
Dynamic Pruning

function FTA-Forward-Pruning (M, Tle)
1   CS := ∅;
2   M := Extend(M, R°);
3   Reach := I ∩ (q = f);
4   Front := I ∩ (q = f);
5   while (Front ≠ ∅) do
6      CS := CS ∪ Proj(q, Reach ∩ Tle);
7         temp := Reach;
8         Reach := Reach ∪
9            fwd_img(M, Front);
10        Front := Reach \ temp;
11      end while;
12   MCS := Minimize(CS);
13   return Map_q→f(MCS);

At each iteration, compute a partial set of cut sets
Use the partial set to prune non-minimal config.
in the search space
Dynamic Pruning

function FTA-Forward-Pruning ($\mathcal{M}$, Tle)
1 $CS := \emptyset$;
2 $\mathcal{M} := \text{Extend}(\mathcal{M}, \mathcal{R})$;
3 $\text{Reach} := \mathcal{I} \cap (\mathcal{g} = \mathcal{f})$;
4 $\text{Front} := \mathcal{I} \cap (\mathcal{g} = \mathcal{f})$;
5 \textbf{while} ($\text{Front} \neq \emptyset$) \textbf{do}
6 $CS := CS \cup \text{Proj}(\mathcal{g}, \text{Reach} \cap \text{Tle})$;
7 $\text{temp} := \text{Reach}$;
8 $\text{Reach} := \text{Reach} \cup$
9 $\text{fwd\_img}(\mathcal{M}, \text{Front})$;
10 $\text{Front} := \text{Reach} \setminus \text{temp}$;
11 $\text{Front} := \text{Front} \setminus \text{Widen}(CS)$;
12 \textbf{end while};
13 $\text{MCS} := \text{Minimize}(CS)$;
14 \textbf{return} $\text{Map}_{\mathcal{g} \rightarrow \mathcal{f}}(\text{MCS})$;

At each iteration, compute a partial set of cut sets
Use the partial set to prune non-minimal config.
in the search space
Dynamic Pruning

function FTA-Forward-Pruning (\(M, Tle\))
1 \(CS := \emptyset;\)
2 \(M := \text{Extend}(M, R^o);\)
3 \(Reach := I \cap (q = f);\)
4 \(Front := I \cap (q = 1);\)
5 while (Front \(\neq \emptyset\)) do
6 \(CS := CS \cup \text{Proj}(q, Reach \cap Tle);\)
7 \(temp := Reach;\)
8 \(Reach := Reach \cup \text{fwd_img}(M, Front);\)
9 \(Front := Reach \setminus temp;\)
10 \(Front := Front \setminus \text{Widen}(CS);\)
11 end while;
12 \(MCS := \text{Minimize}(CS);\)
13 return Map_{q \rightarrow f}(MCS);

At each iteration, compute a partial set of cut sets
Use the partial set to prune non-minimal config. in the search space
Dynamic Pruning

function FTA-Forward-Pruning (\(\mathcal{M}, Tle\))
1 \(CS := \emptyset;\)
2 \(\mathcal{M} := \text{Extend}(\mathcal{M}, \mathcal{R}^o);\)
3 \(\text{Reach} := \mathcal{I} \cap (\mathcal{O} = \mathcal{I});\)
4 \(\text{Front} := \mathcal{I} \cap (\mathcal{O} = \mathcal{I});\)
5 \(\text{while } (\text{Front} \neq \emptyset) \text{ do} \)
6 \(CS := CS \cup \text{Proj}(\mathcal{O}, \text{Reach} \cap Tle);\)
7 \(\text{temp} := \text{Reach};\)
8 \(\text{Reach} := \text{Reach} \cup\)
9 \(\text{fwd_img}(\mathcal{M}, \text{Front});\)
10 \(\text{Front} := \text{Reach} \setminus \text{temp};\)
11 \(\text{Front} := \text{Front} \setminus \text{Widen}(CS);\)
12 \(MCS := \text{Minimize}(CS);\)
13 \(\text{return } \text{Map}_{\mathcal{O} \rightarrow I}(MCS);\)

At each iteration, compute a partial set of cut sets
Use the partial set to prune non-minimal config.
in the search space
Dynamic Pruning

function FTA-Forward-Pruning \((\mathcal{M}, \text{Tle})\)

1. \(CS := \emptyset;\)
2. \(\mathcal{M} := \text{Extend}(\mathcal{M}, \mathcal{R}^\circ);\)
3. \(\text{Reach} := \mathcal{I} \cap (\mathcal{Q} = \mathcal{F});\)
4. \(\text{Front} := \mathcal{I} \cap (\mathcal{Q} = \mathcal{F});\)
5. \(\textbf{while} (\text{Front} \neq \emptyset) \textbf{do}\)
6. \(CS := CS \cup \text{Proj}(\mathcal{Q}, \text{Reach} \cap \text{Tle});\)
7. \(\text{temp} := \text{Reach};\)
8. \(\text{Reach} := \text{Reach} \cup \text{fwd\_img}(\mathcal{M}, \text{Front});\)
9. \(\text{Front} := \text{Reach}\setminus\text{temp};\)
10. \(\text{Front} := \text{Front}\setminus\text{Widen}(CS);\)
11. \(\textbf{end while};\)
12. \(MCS := \text{Minimize}(CS);\)
13. \(\text{return } Map_{\mathcal{Q} \rightarrow \mathcal{F}}(MCS);\)

At each iteration, compute a partial set of cut sets
Use the partial set to prune non-minimal config.
in the search space
Dynamic Pruning

function FTA-Forward-Pruning \((M, Tle)\)
1. \(CS := \emptyset;\)
2. \(M := \text{Extend}(M, \mathcal{R}^o);\)
3. \(\text{Reach} := I \cap (q = \mathfrak{f});\)
4. \(\text{Front} := I \cap (q = \mathfrak{f});\)
5. while \((\text{Front} \neq \emptyset)\) do
6. \(CS := CS \cup \text{Proj}(q, \text{Reach} \cap Tle);\)
7. \(\text{temp} := \text{Reach};\)
8. \(\text{Reach} := \text{Reach} \cup \text{fwd_{img}}(M, \text{Front});\)
9. \(\text{Front} := \text{Reach} \setminus \text{temp};\)
10. \(\text{Front} := \text{Front} \setminus \text{Widen}(CS);\)
end while;
12. \(MCS := \text{Minimize}(CS);\)
13. return \(\text{Map}_{o \rightarrow f}(MCS);\)

At each iteration, compute a partial set of cut sets
Use the partial set to prune non-minimal config.
in the search space

R. Banach, School of Computer Science, University of Manchester, UK
M. Bozzano, Fondazione Bruno Kessler, FBK-IRST, Trento, Italy
Dynamic Pruning

function FTA-Forward-Pruning (M, Tle)
1  CS := Ø;
2  M := Extend(M, R°);
3  Reach := I \cap (q = f);
4  Front := I \cap (q = f);
5  while (Front ≠ Ø) do
6      CS := CS \cup Proj(q, Reach \cap Tle);
7          temp := Reach;
8          Reach := Reach \cup fwd_img(M, Front);
9      Front := Reach \setminus temp;
10     Front := Front \setminus Widen(CS);
11  end while;
12  MCS := Minimize(CS);
13  return Map_{q→f}(MCS);

At each iteration, compute a partial set of cut sets
Use the partial set to prune non-minimal config.
in the search space
Dynamic Pruning

function FTA-Forward-Pruning ($M, Tle$)
1 $CS := \emptyset$;
2 $M := \text{Extend}(M, R^\circ)$;
3 $\text{Reach} := I \cap (\omega = f)$;
4 $\text{Front} := I \cap (\omega = \overline{f})$;
5 while ($\text{Front} \neq \emptyset$) do
6 $CS := CS \cup \text{Proj}(\omega, \text{Reach} \cap Tle)$;
7 $\text{temp} := \text{Reach}$;
8 $\text{Reach} := \text{Reach} \cup$
9 $\text{fwd\_img}(M, \text{Front})$;
10 $\text{Front} := \text{Reach} \setminus \text{temp}$;
11 $\text{Front} := \text{Front} \setminus \text{Widen}(CS)$;
12 end while;
13 $MCS := \text{Minimize}(CS)$;
14 return $\text{Map}_{\omega \rightarrow f}(MCS)$;

At each iteration, compute a partial set of cut sets
Use the partial set to prune non-minimal config.
in the search space
Dynamic Pruning

function FTA-Forward-Pruning \((M, \text{Tle})\)

1. \(CS := \emptyset;\)
2. \(M := \text{Extend}(M, R^\circ);\)
3. \(\text{Reach} := \mathcal{I} \cap (q = f);\)
4. \(\text{Front} := \mathcal{I} \cap (q = f);\)
5. \(\text{while} (\text{Front} \neq \emptyset) \text{ do}\)
6. \(CS := CS \cup \text{Proj}(q, \text{Reach} \cap \text{Tle});\)
7. \(\text{temp} := \text{Reach};\)
8. \(\text{Reach} := \text{Reach} \cup \text{fwd_img}(M, \text{Front});\)
9. \(\text{Front} := \text{Reach} \setminus \text{temp};\)
10. \(\text{Front} := \text{Front} \setminus \text{Widen}(CS);\)
11. \(\text{end while};\)
12. \(MCS := \text{Minimize}(CS);\)
13. \(\text{return } \text{Map}_{g \to f}(MCS);\)

At each iteration, compute a partial set of cut sets

Use the partial set to prune non-minimal config.
in the search space
Dynamic Pruning

function FTA-Forward-Pruning \((\mathcal{M}, Tle)\)
1. \(CS := \emptyset;\)
2. \(\mathcal{M} := \text{Extend}(\mathcal{M}, \mathcal{R}^o);\)
3. \(\text{Reach} := \mathcal{I} \cap (\mathcal{A} = \mathcal{f});\)
4. \(\text{Front} := \mathcal{I} \cap (\mathcal{A} = \mathcal{f});\)
5. while (\text{Front} \neq \emptyset) do
6.   \(CS := CS \cup \text{Proj}(\mathcal{A}, \text{Reach} \cap Tle);\)
7.   \(\text{temp} := \text{Reach};\)
8.   \(\text{Reach} := \text{Reach} \cup \text{fwd imaging}(\mathcal{M}, \text{Front});\)
9.   \(\text{Front} := \text{Reach} \setminus \text{temp};\)
10. \(\text{Front} := \text{Front} \setminus \text{Widen}(CS);\)
end while:
12. \(MCS := \text{Minimize}(CS);\)
13. return \(\text{Map}_{\mathcal{A} \rightarrow \mathcal{f}}(MCS);\)

At each iteration, compute a partial set of cut sets
Use the partial set to prune non-minimal config. in the search space

R. Banach, School of Computer Science, University of Manchester, UK
M. Bozzano, Fondazione Bruno Kessler, FBK-IRST, Trento, Italy
12. Retrenchment and Model Checking Compared.

We compare:

- The Retrenchment-based FT generation algorithm.
- The BDD-based backward FT generation algorithm.
  (with Dynamic Cone of Influence and Dynamic Pruning)

In the general case of feedback circuits with time delays.

Strong similarities:
- Most important: backward resolution, i.e. start from the TLE.

But – several differences:
- Related to: system decomposition, search strategy, etc.
- Implementation-level but also theoretical differences.

Discussion: how to reconcile retrenchment with model checking.
System Decomposition

Retrenchment-based: decomposition based on system structure.
BDD-based: decomposition based on time delays.

Consequences:
• No difference if unit delays between every block.
  (e.g., adders and fanouts in the circuit example).
• In the purely combinational case – no delays:
  BDD-based flattens the system – monolithic transition relation.

Reconciliation:
• Not a huge difference: BDD-based could be instructed to take system structure
  into account, or use “hybrid” strategies.
Search Strategy

Retrenchment-based: non-deterministic, data dependency driven, search.
- Depth-first search illustrated here,
  although simplified by the use of angelic non-determinism theoretically.

BDD-based: breadth-first search.
- Each step decomposing one layer of the composition.
- Efficiency of breadth-first search relies on the BDD package.
- Dynamic pruning introducing controlled depth-first aspects in the search.

Reconciliation:
- Not a huge difference: search strategy in BDD-based is flexible.
- Possibly introducing further depth-first aspects in BDD-based,
  e.g. descend first in branches with a lower number of faults, and then prune.
- Mostly an implementation detail.
Minimisation Rules

Retrenchment-based: minimisation rules to prune the search tree on the fly.

BDD-based: minimisation rules mimicked by the internals of the algorithm or by the BDD package.

- Discarding non-needed subtrees $\rightarrow$ BDD package + DCOI reduction rules.
- Discarding subtrees at input-insensitive faults $\rightarrow$ BDD package.
- Discarding locally subsumed expressions $\rightarrow$ dynamic pruning.
- Subsumption checking at the subsystem level $\rightarrow$ dynamic pruning.

Reconciliation:

- Not needed.
Timing and Feedback

Retrenchment-based: deals with time delays explicitly.
- Time information fully recorded.
- Different definitions of minimality may be used to turn a RT into a FT.

BDD-based: deals with time delays tacitly.
- Time information is not recorded, temporal details abstracted away,
  same states with different time delays are identified.
- Directly generates the fault trees representing the minimal cut sets.
  (where definition of minimality abstracts away from time)

Reconciliation:
- Soundness: we get the same results in both cases if we abstract away from time.
- Possibly introducing handling of timing information in BDD-based – but in practice may have an impact on performance. Need to deal with sets of traces rather than sets of states, in a controlled way.
Initial States and Cold-Start Failures

Retrenchment-based: uses appropriate truncation of the detailed FT.
- Minimisation performed independently of initialisation.

BDD-based: truncation performed on-the-fly, reachability check built in.
- Tight coupling of initialisation and minimisation.
- Minimisation may interact with timing abstraction:
  - It may discard states that have been identified because of timing abstraction.
  - It may discard hot-running scenarios in favour of cold-start ones.
  - Focus on computation of MCSs, rather than fault trees.

Reconciliation:
- It is possible to rule out cold-start scenarios in BDD-based, if desired.
- Possibly introducing further guidance in BDD-based to deal with hot-running and cold-start failures.
Conclusions

**Retrenchment-based**: an idealised specification of a FT generation algorithm.
  Can move it closer to the BDD-based algorithm by carefully *forgetting* details.

**BDD-based**: an implementation that does not completely conform to it.
  Can move it closer to the Retrenchment-based ‘ideal’ by *including* more details
  … but you have to watch performance in practice.