Software Engineering with Formal Methods: The Development of a Storm Surge Barrier Control System
Revisiting Seven Myths of Formal Methods

JAN TREMANS tretmans@cs.utwente.nl
University of Twente, Department of Computer Science, Formal Methods and Tools Research Group, P.O. Box 217, 7500 AE Enschede, The Netherlands

KLAAS WIJBRANS* klaas.wijbrans@cmg.nl
MICHEL CHAUDRON† m.r.v.chaudron@tue.nl
CMG Public Sector B.V., Division Advanced Technology, P.O. Box 187, 2501 CD The Hague, The Netherlands

Received November 30, 1999; Accepted March 15, 2000

Abstract. This paper discusses the use of formal methods in the development of the control system for the Maeslant Kering. The Maeslant Kering is the movable dam which has to protect Rotterdam from floodings while, at (almost) the same time, not restricting ship traffic to the port of Rotterdam. The control system, called Bos, completely autonomously decides about closing and opening of the barrier and, when necessary, also performs these tasks without human intervention. Bos is a safety-critical software system of the highest Safety Integrity Level according to IEC 61508. One of the reliability increasing techniques used during its development is formal methods. This paper reports experiences obtained from using formal methods in the development of Bos. These experiences are presented in the context of Hall’s famous “Seven Myths of Formal Methods”.

Keywords: industrial application of formal methods
Formal methods in this course

» Introduction to formal models
  » Labelled transition systems (LTSs)
  » Timed Automata (TA)

» Introduction to formal temporal logics
  » Linear Temporal Logic (LTL)

» Application using UPPAAL
  » TA-like models
Further courses at the master level

- System validation
- Process algebra
- Algorithms for model-checking
- Hardware verification
- Program verification techniques
- Verification of security protocols
- Automated reasoning
- Proving with computer assistance

- Mainly active in the Systems and Software Science streams.
References

» Two good references
» You are welcome to consult them, but this is optional
Program for today

» Labelled transitions system
  » definition
  » executions
  » paths

» Linear time behaviours over paths
  » Linear Temporal Logic
Consider a simple tea/coffee machine:

- accept “coins”
- after one coin, user may press “Start” and get tea
- after two coins, user may press “Start” and get coffee

Objective:
- formally reasons about the machine behaviour
- precisely express requirements
- obtain models that formally possess the requirements
Three ingredients

1. Formal model
2. Temporal Logic
3. Algorithms

In this course, we will introduce 1 and 2.

Details of 1 and 2 as well as 3 fall outside the scope of this course.
Let’s write a few requirements for the coffee machine:

- R1: ...

- R2: ...
Let’s write a few requirements for the coffee machine:

- R1: After inserting a coin and pressing the start button, the system shall produce tea.
- R2: After inserting two coins and pressing the start button, the system shall produce coffee.
Ingredient 1: Formal models

Basic idea: states and transition between states.

Model of the coffee machine?
Labelled transition systems (LTSs)

Formal representation of systems.

States (bullets) and transitions (arrows).

LTSs used to give formal semantics
- to sequence diagrams
- to state machine diagrams
- to process algebra
- to parallel programs
- etc
Labelled transition systems (LTSs)

Formal representation of systems.

States (bullets) and transitions (arrow).

LTSs are a formal representation of:
- the states of the system
- the stepwise behaviour
- the initial states

Additional information about
- communication (actions)
- state properties (atomic proposition)
LTS definition

tuple (S, Act, →, I, AP, L) where

S is a set of states
Act is a set of actions
→ ⊆ S × Act × S is a transition relation
I ⊆ S is a set of initial states
AP is a set of atomic propositions
L : S → 2^AP is a state labelling function

An LTS is finite if S, Act, and AP are finite.

s \xrightarrow{\alpha} s' is the short notation for (s, \alpha, s') ∈ →
\( S = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6\} \)
\( I = \{s_0\} \)
\( AP = \{\} \)
\( L = \text{forall } s, \; L(s) = \{\} \)
\( \text{Act} = \{\text{coin, start, tea, coffee}\} \)
Internal action: Represent some internal activity.

\[ \tau \]

\[ S = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6\} \]

\[ I = \{s_0\} \]

\[ AP = \{\} \]

\[ L = \text{forall } s, L(s) = \{\} \]

\[ \text{Act} = \{\text{coin, start, tea, coffee}\} \cup \{\tau\} \]
Atomic Propositions.

Chosen depending on the characteristics of interest.

Example: The machine only delivers coffee after two coins have been inserted.

\[ S = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6\} \]
\[ I = \{s_0\} \]
\[ AP = \{\} \]
\[ L = \forall s, L(s) = \{\} \]
\[ Act = \{\text{coin, start, tea, coffee}\} \cup \{\tau\} \]
Context: our goal is to formalise our requirements:

- R1: After inserting a coin and pressing the start button, the system shall produce tea.
- R2: After inserting two coins and pressing the start button, the system shall produce coffee.

We will select/create set AP accordingly.
Atomic Propositions.

Chosen depending on the characteristics of interest.

Example: The machine only delivers coffee after two coins have been inserted.

$S = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6\}$

$I = \{s_0\}$

$AP = \{\text{coin2}, \text{ServingCoffee}\}$

$L = \forall s, L(s) = \{\}$

$Act = \{\text{coin, start, tea, coffee}\} \cup \{\tau\}$

$\text{coin2}$: is true when two coins have been inserted

$\text{ServingCoffee}$: is true when coffee is being served.
Atomic Propositions.

Chosen depending on the characteristics of interest.

Example: The machine only delivers coffee after two coins have been inserted.

\[ S = \{s0, s1, s2, s3, s4, s5, s6\} \]
\[ I = \{s0\} \]
\[ AP = \{coin2, ServingCoffee\} \]
\[ L: L(s4) = \{coin2\}, L(s5) = \{coin2, ServingCoffee\}, L(s) = \{\} \text{ otherwise} \]
\[ Act = \{coin, start, tea, coffee\} \cup \{\tau\} \]

\textit{coin2}: is true when two coins have been inserted

\textit{ServingCoffee}: is true when coffee is being served.
Executions of LTSs

Possible behaviours result from:

WHILE \( s \) is non-terminal DO

select non-deterministically a transition \( s \xrightarrow{\alpha} s' \)

execute action \( \alpha \) and update \( s := s' \)

Execution: “transition sequences”

\[
\rho = s_0\alpha_0s_1\alpha_1s_2\alpha_2s_3\ldots
\]

such that \( s_i \xrightarrow{\alpha_i} s_{i+1}, \forall 0 \leq i \)

N.B.:
- executions potentially infinite
- executions start with the initial state
- executions always end with a terminal state
- or are infinite

Examples:

\[
\rho_0 = s_0 \text{ coin } s_1 \text{ coin } s_4 \text{ start } s_5 \text{ coffee } s_6 \tau s_0\ldots
\]

\[
\rho_1 = s_0 \text{ coin } s_1 \text{ start } s_2 \text{ tea } s_3 \tau s_0\ldots
\]
Pre and post states

Successor states of state $s$:

$$Post(s, \alpha) = \{s'|s \xrightarrow{\alpha} s'\}$$

$$Post(s) = \bigcup_{\alpha \in Act} Post(s, \alpha)$$

Predecessor states of state $s$:

$$Pre(s, \alpha) = \{s'|s' \xrightarrow{\alpha} s\}$$

$$Pre(s) = \bigcup_{\alpha \in Act} Pre(s, \alpha)$$

Examples:

$Pre(s_4) = \{s_1\}$

$$Post(s_1) = \{s_2, s_4\}$$
Reachable states

Set of states **reachable** from an initial state through some execution.

A state \( s \) is reachable if there exists a finite execution prefix ending in \( s \):

\[
s_0 \alpha_0 s_1 \alpha_1 \ldots \alpha_n s_n = s
\]

The set of reachable states of an LTS \( T \) is noted:

\[
\text{Reach}(T)
\]
Non-determinism
Non-determinism - definition

We will say that transition system $T$ is **deterministic** if

\[ |I| = 1 \quad \text{It has exactly one initial state.} \]

\[ \forall s \in S, \forall \alpha \in \text{Act. } |Post(s, \alpha)| \leq 1 \quad \text{For all actions and states, it has at most one successor.} \]
Paths and associated notations

A path is the projection of executions to states.

A path is a (possibly infinite) sequence of states

\[ s_0s_1s_2\ldots \text{ such that } \forall i > 0. s_i \in Post(s_{i-1}) \]

Notations for paths:

\[ \pi = s_0s_1s_2\ldots \text{ An infinite path.} \]
\[ \pi[i] = s_i \text{ The } ith \text{ state of a path.} \]
\[ \pi[..i] = s_0s_1\ldots s_i \text{ Finite prefix up to position } i. \]
\[ \pi[i..] = s_is_{i+1}\ldots \text{ Suffix from position } i \]
Paths - Examples

Executions:
\[ \rho_0 = s_0 \text{ coin } s_1 \text{ coin } s_4 \text{ start } s_5 \text{ coffee } s_6 \tau s_0 \ldots \]
\[ \rho_1 = s_0 \text{ coin } s_1 \text{ start } s_2 \text{ tea } s_3 \tau s_0 \ldots \]

Corresponding paths:
\[ \pi_0 = s_0s_1s_4s_5s_6s_0 \ldots \]
\[ \pi_1 = s_0s_1s_2s_3s_0 \ldots \]
Program for today

» Labelled transitions system
  » definition
  » executions
  » paths

» Linear time behaviours over paths
  » Linear Temporal Logic
Linear Temporal Logic

» Reason about atomic propositions and paths
  » When is a proposition true?

» Discrete notion of time
  » next time = next position in a path

» Consider infinite paths only

» Two important operators:
  » Next - notation X
  » Until - notion U
A valid LTL formula has the following syntax:

\[ \varphi ::= a \mid \varphi_1 \land \varphi_2 \mid \neg \varphi \mid X\varphi \mid \varphi_1 U \varphi_2 \]

where
- \.. is an atomic proposition \( a \in AP \)
- \.. reads “next”
- \.. reads “until”
Boolean connectives have their usual truth value.

\[ \pi \models a \quad \text{if and only if} \quad a \in L(\pi[0]) \]

\[ \pi \models \neg a \quad \text{if and only if} \quad a \notin L(\pi[0]) \]

\[ \pi \models a \land b \quad \text{if and only if} \quad \pi \models a \text{ and } \pi \models b \]

Examples:

\[ \pi = s_0 s_1 s_4 s_5 s_6 s_0 \ldots \]

\[ \pi[2..] \models \text{coin2} \]

\[ \pi[3..] \models \text{coin2} \land \text{ServingCoffee} \]

\[ \pi[1..] \models \neg \text{coin2} \]
LTL semantics — Next “X”

Next means “next time”, so at the next position in the path.

$$\pi \models X\varphi \quad \text{if and only if} \quad \pi[1..] \models \varphi$$

Examples:

$$\pi = s_0s_1s_4s_5s_6s_0...$$

$$\pi \models X\neg\text{coin}2$$

$$\pi[2..] \models X(\text{coin}2 \land \text{ServingCoffee})$$

$$\pi[3..] \models X(\neg\text{coin}2 \land \neg\text{ServingCoffee})$$

$$\pi \models X^3(\text{coin}2 \land \text{ServingCoffee})$$
LTL semantics — Until “U”

\( \varphi_1 \text{ Until } \varphi_2 \) is defined by:

\[
\pi \models \varphi_1 \mathbf{U} \varphi_2
\]

if and only if

\[
\exists j \geq 0. \pi[j..] \models \varphi_2 \text{ and } \forall 0 \leq i < j. \pi[i..] \models \varphi_1
\]

\( \varphi_2 \) must hold for some position

\( \varphi_1 \) holds at all previous positions

Examples:

\[
\pi = s_0s_1s_4s_5s_6s_0...
\]

\[
\pi \models \neg \text{coin2} \ \mathbf{U} \ \text{coin2}
\]

\[
\pi \models \neg \text{coin2} \ \mathbf{U} \ (X \ \text{coin2})
\]

\[
\pi \models \neg \text{coin2} \ \mathbf{U} \ X^2(\text{coin2} \land \text{ServingCoffee})
\]
Derived operators — more Boolean connectives

\[
\text{True} = \varphi \lor \neg \varphi
\]

\[
\text{False} = \neg \text{True}
\]

\[
\varphi_1 \lor \varphi_2 = \neg (\neg \varphi_1 \land \neg \varphi_2)
\]

\[
\varphi_1 \rightarrow \varphi_2 = \neg \varphi_1 \lor \varphi_2
\]
Derived operator — Eventually “F”

Eventually means “true in the future”

\[ F\varphi = \text{True} \cup \varphi \]

This expands to the following semantics:

\[ \exists i \geq 0. \pi[i..] \models \varphi \]

Examples:

\[ \pi = s_0 s_1 s_4 s_5 s_6 s_0 \ldots \]

\[ \pi \models F \text{ServingCoffee} \]

\[ \pi \models F (X \text{ServingCoffee}) \]

\[ \pi \models F (\neg \text{coin2} \cup (X \text{ServingCoffee})) \]
Derived operator — Globally “G”

Globally means “always true”

\[ G\varphi = \neg F \neg \varphi \]

It expands to the following

\[ \forall i \geq 0. \pi[i..] \models \varphi \]
So, far defined semantics for formula over paths.

In LTL, a formula is true in a given state if it holds on all paths starting from that state.

\[ s \models \varphi \iff \forall \pi \in Paths(s). \pi \models \varphi \]

where \( Paths(s) \) denote the set of paths starting in \( s \).

Examples:

\[ F \text{ ServingCoffee} \]

For which state is it true?
Formula true in a state

» So, far defined semantics for formula over paths.

» In LTL, a formula is true in a given state if it holds on all paths starting from that state

\[ s \models \varphi \quad \text{if and only if} \quad \forall \pi \in \text{Paths}(s). \pi \models \varphi \]

where \( \text{Paths}(s) \) denote the set of paths starting in \( s \).

Examples:

\[ F \text{ ServingCoffee} \]

\[ s_4 \models F \text{ ServingCoffee} \]

\[ s_0 \not\models F \text{ ServingCoffee} \]
Formula true for an LTS

An LTS satisfies a formula if the formula holds in all initial states.

\[ \text{LTS} \models \varphi \quad \text{if and only if} \quad \forall s_0 \in I.s_0 \models \varphi \]
Back to the initial requirement

» How do we formulate our original requirements?
Let’s write a few requirements:

- **R1**: After inserting a coin and pressing the start button, the system shall produce tea.
- **R2**: After inserting two coins and pressing the start button, the system shall produce coffee.
- **R1:** After inserting a coin and pressing the start button, the system shall produce tea.

We need to define some propositions:

- **coin1**  
  1 coin has been asserted

- **start**  
  Start button has been activated

- **ServingTea**  
  Tea is about to be served.

We need to define the labelling function:

- \( L(s_1) = \{ \text{coin1} \} \)
- \( L(s_2) = \{ \text{coin1, start, ServingTea} \} \)

We can then write our requirement:

\[
 r_1 = \mathbf{G} ((\text{coin1} \land \text{start}) \rightarrow \text{ServingTea}) \\
 r_1 = \mathbf{G} (\neg\text{coin1} \lor \neg\text{start} \lor \text{ServingTea})
\]
- **R2**: After inserting two coins and pressing the start button, the system shall produce coffee.

We need to define some additional propositions:

- \(coin2\) \quad 2 coin has been asserted
- \(ServingCoffee\) \quad Coffee is about to be served.

We need to refine the labelling function:

\[
L(s_4) = \{coin2\} \\
L(s_5) = \{coin2, start, ServingCoffee\}
\]

We can then write our requirement:

\[
r_2 = G((coin2 \land start) \rightarrow ServingCoffee) \\
r_2 = G(\neg coin2 \lor \neg start \lor ServingCoffee)
\]
Some more LTL formulas examples

» At the next clock cycle, the request signal must be high.

» The request signal must be high unit the acknowledge is high.

» The arbiter always grants at most one request.

» The elevator should never travel when the doors are open.
Some more LTL formulas examples

» At the next clock cycle, the request signal must be high.

\[ X \ signal \]

» The request signal must be high until the acknowledge is high.

\[ req \ U \ ack \]

» The arbiter always grants at most one request.

\[ G \ \neg (req1 \land req2) \]

» The elevator should never travel when the doors are open.

\[ G \ \neg (moving \land doorOpen) \]
Some typical of LTL properties

» In general:
  » Safety: something bad will never happen \( G \neg \varphi \)
  » Liveness: something will eventually happen \( F \varphi \)

» Request-response \( G (\varphi_1 \rightarrow F \varphi_2) \)
Checking LTL formulas

» Details out-of-score for this course.
  » see master courses for more details

» One possible way:
  » exhaustive state-space exploration
  » explore all states and check that the property is true
  » works for finite-state systems
  » state-explosion problems