Formal Methods

Markus Roggenbach

Feburary 2021

Underlying Question: How Can We Trust Software?

Cyberpunk 2077 bugs: The very best of the worst

`https://www.tomsguide.com/uk/news/cyberpunk-2077-bugs`

TOP 5 Software Failures of 2018–2019

- Facebook's apps outage
- CPUs flaw
- Crashed lunar lander
- British Airways glitch
- Self-driving killer car

Overview of the next two weeks

Overview

Formal Methods are one means in Software Engineering that can help ensure that a computer system meets its requirements.

- Formal Methods (1 lecture)
- The Process Algebra CSP (2 lecures, 1 lab)
- Verifying a security protocol with CSP (1 lecture)

Please install the tool FDR4 on your computers:
`https://cocotec.io/fdr/`

Material from MR's FMSE book (to appear)

Springer

springer.com

M. Roggenbach, A. Cerone, B.-H. Schlingloff, G. Schneider, S.A. Shaikh

Formal Methods for Software Engineering

Languages, Methods, Application Domains

- Authors are all highly experienced educators and researchers
- Book suitable for graduate and undergraduate courses in Software Engineering
- Explains foundations, introduces specification and testing methods, and then explores application domains

This is a graduate-level introduction to formal methods. The first part presents two formal languages: logic, in various forms, and Communicating Sequential Process (CSP) as a process algebra. The second part offers specification and testing methods for formal development of software. Building on the foundations from the first part, the reader is allowed to embrace methods for practical applications. The reader will find the examples cutting across chapters valuable for this purpose. The final section takes the reader further into application domains.

1st ed. 2020, X, 440 p. 35 illus., 10 illus. in color.

The relevant book chapters

Chapter 1
Formal Methods

Gerardo Schneider[1] and Markus Roggenbach[2] and Bernd-Holger Schlingloff[3]

Chapter 3
The process algebra CSP

Antonio Cerone and Markus Roggenbach and Siraj Ahmed Shaikh

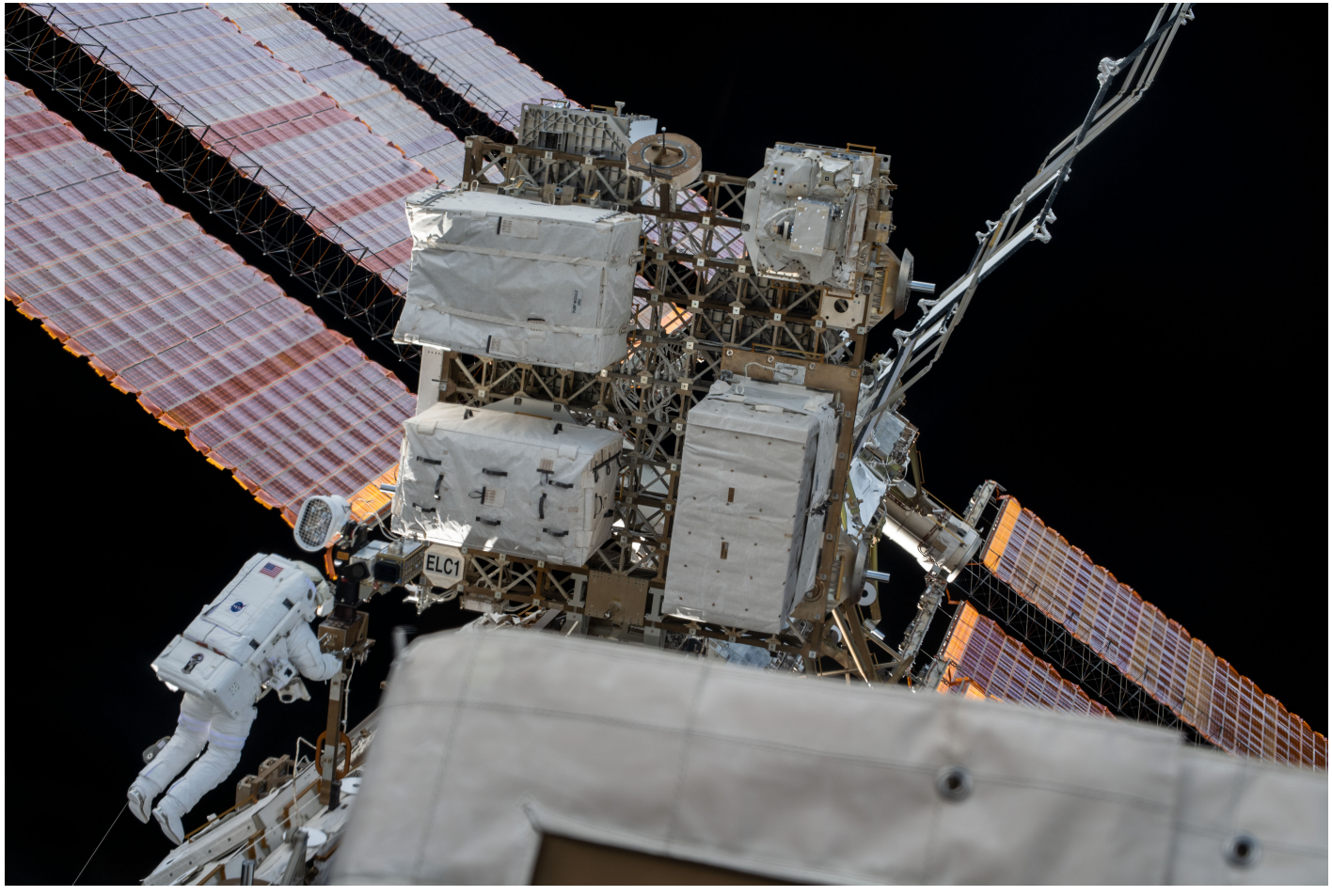
Chapter 8
Formal Verification of Security Protocols

Markus Roggenbach, Siraj Ahmed Shaikh, and Hoang Nga Nguyen

will be on Canvas.

FMs (at) work: ISS

Fault Tolerant Computer (FTC) of the ISS



Byzantine agreement protocol

ISS-FTC:

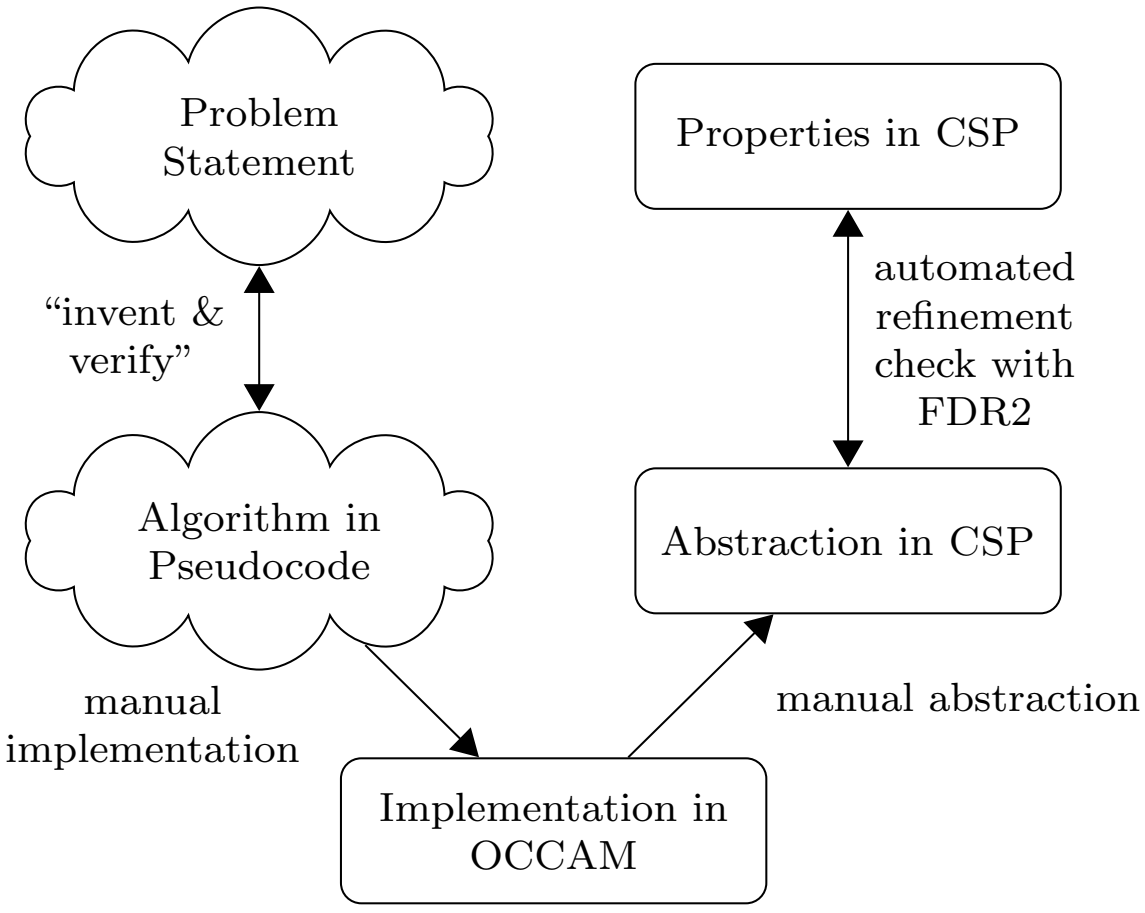
- four identical interconnected hardware boards,
- which perform essentially the same computation.

Faulty board:

- can generate wrong messages &
- modify messages of the other (correct) boards

Solution: Implement a Byzantine agreement protocol as provided in the literature.

Verification approach by Buth et al. (1997-99)



Published findings by Buth et al. (1997-99)

- "seven deadlock situations were uncovered", and
- "about five livelocks were detected"

in the software of the FTC fault management layer.

FMs (at) work: string replacement

Motivating problem

Task: Remove all comments from an HTML document.
Info: HTML comments begin with "<--", end with "-->".

Editors offer replacement based on *regular expressions*:
the symbol '*' serves as a wildcard
$\rightsquigarrow$ Replacing "<--*-->" by an empty string yields the desired result.

How do different editors work with the wildcard sign '*'?

Word:

- replace '*' by 'x' in 'abc' results in 'xxxx'

Emacs:

- replace '*' by 'x' in 'abc' results in 'xxx'
- replace '.*' by 'x' in 'abc' results in 'x'

Question: What is / should the semantics of replacement be?

Syntax

Syntactically, each Formal Method deals with objects from a formal language, usually defined by a grammar.

Example (Grammar of regular expressions)

Given an alphabet $\mathcal{A}$, the language of regular expressions is given by the following grammar:

- every letter from the alphabet is a regular expression.
- λ is a regular expression.
- if φ and ψ are regular expressions, then $(\varphi\ \psi)$ and $(\varphi + \psi)$ are regular expressions.
- if φ is a regular expression, then φ^* is a regular expression.

The same definition can be written in so-called Backus-Naur-Form:

$$\begin{aligned} Regexp_{\mathcal{A}} \quad ::= \quad & a \in \mathcal{A} \quad | \quad \lambda \quad | \quad (Regexp_{\mathcal{A}}\ Regexp_{\mathcal{A}}) \quad | \\ & (Regexp_{\mathcal{A}} + Regexp_{\mathcal{A}}) \quad | \quad Regexp_{\mathcal{A}}^{*} \end{aligned}$$

Semantics

In Formal Methods, a formal language comes with a formal semantics which explains the meaning of the syntactical objects by interpreting it in some domain.

Example (Denotational semantics of regular expressions)

- $[\![a]\!] \triangleq \{a\}$ for any $a \in \mathcal{A}$.
- $[\![\lambda]\!] \triangleq \{\}$.
- $[\![(\varphi\ \psi)]\!] \triangleq \{xy \mid x \in [\![\varphi]\!], y \in [\![\psi]\!]\}$.
- $[\![(\varphi + \psi)]\!] \triangleq [\![\varphi]\!] \cup [\![\psi]\!]$.
- $[\![\varphi^*]\!] \triangleq \{x_1 \ldots x_n \mid n \geq 0,$ and for all $i \leq n$, $x_i \in [\![\varphi]\!]\}$.

The semantic domain of regular expressions are sets of strings.

Hint: other 'popular' semantics are operational semantics and axiomatic semantics.

Method

Formal Methods come with some algorithms/procedures which describe how syntactic objects can be manipulated.

Example (Regular replacement)

$\delta = \gamma[\alpha := \beta]$ – i.e., replace regular expression α with word β on a word γ results in δ if one of the following holds:

1. Either there exist γ_1, γ_2 and γ_3 such that
   - 1.1 $\gamma = \gamma_1\gamma_2\gamma_3$,
   - 1.2 $\gamma_2 \in [\![\alpha]\!] \setminus [\![\varepsilon]\!]$,
   - 1.3 γ_1 is of minimal length,
   - 1.4 γ_2 is of maximal length,

   and $\delta = \gamma_1\beta(\gamma_3[\alpha := \beta])$, or
2. there are no γ_1, γ_2 and γ_3 satisfying the above 1.1–1.4., and $\delta = \gamma$.

Note: $\varepsilon \triangleq \lambda^*$ ("the empty word").

Working definition: Formal Method

A Formal Method $\mathbb{M}$ consists of three components:

- syntax,
- semantics, and
- method.

syntax: precise description of the form of objects (strings or graphs) belonging to $\mathbb{M}$.
semantics: describes the "meaning" of the syntactic objects of $\mathbb{M}$, in general by a mapping into some mathematical structure.
method: describes algorithmic ways of transforming syntactic objects, in order to gain some insight about them.

Use of Formal Methods

FMs as means to make descriptions precise

The inherent ambiguity and lack of precision of natural language makes the realization of requirements, specifications, models written in natural language problematic.

Kamsties et al. provide as examples e.g.:

- The 500 most used words in English have on average 23 meanings.
- "The product shall show the weather for the next 24 hours".
  - ambiguity: "for the next twenty-four hours" – attached "show" or "weather"?

Formal Methods example: regular replacement.

FMs help in different kinds of analysis

The use of *Formal Methods* in Verification and Validation (often abbreviated as V&V) is wide and includes techniques such as static analysis, formal testing, model checking, runtime verification, and theorem proving.
Formal Methods are complementary techniques to *standard ('informal') methods* such as code review, testing and debugging. They increase the confidence in the correctness of the software under consideration.

Formal Methods example: work of Buth et al. in the context of the ISS.

Tools for Formal Methods

Formal Methods need tool support

Software systems are fundamentally different compared to mathematical theories:

1. Numbers of axioms involved.
   - SW: Typical interlocking program written in Ladder Logic consist of 5000 – 10000 lines of code; its formal model has the same number of axioms.
   - Math: Group theory is based on three axioms only.
2. Ownership / interest
   - SW: IP of interlocking program is with rail operator; its verification is of interest only for its developers and certifiers.
   - Math: Group theory is public; theorems of group theory are published in books and journals, taught for educational purposes at univeristies.
3. Change of axiomatic basis
   - SW: every system update requires new verification as the axioms have changed
   - Math: Axioms of group theory have been stable since the early 1830ties.

FM as a ‘tool’ in SE

The current "software quality crisis"

- Late 1960's: "software crisis"
  - Characterized by:

    cost of software $>$ cost of hardware.
- Nowadays: "software quality crisis" (B-H Schlingloff)
  - Characterized by:

    cost of verification and validation $>$ cost of programming

The typical FM challange

Typcial situation:
A method works on 'toy examples' (TRL 1 – 3)
but will it scale to the relevant ones? (TRL 4 - 9)

MR's conjecture:
(founded in experience in a number of technology transfer projects)

As software is human made
it is always possible to find abstractions
that allows formal methods to scale.

TRL: Technology Readiness Levels
see e.g. `https://web.archive.org/web/20171011071816/https://www.innovation.cc/discussion-papers/22_2_3_heder_nasa-to-eu-trl-scale.pdf`