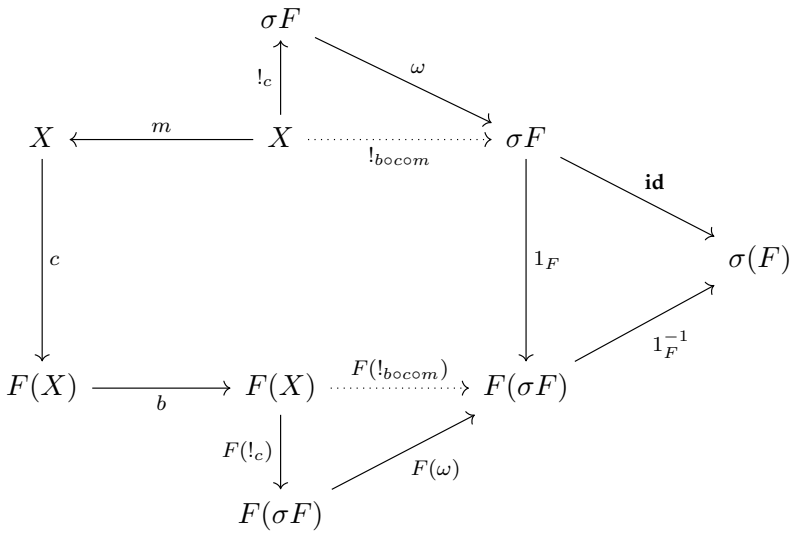
SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN



Latent-Behaviour Analysis

Submitted by

Eric G. ROTHSTEIN MORRIS

Thesis Advisor

Dr. Sudipta CHATTOPADHYAY

Information Systems Technology and Design (ISTD)

A thesis submitted to the Singapore University of Technology and Design in fulfillment of the requirement for the degree of Doctor of Philosophy

2021

PhD Thesis Examination Committee

TEC Chair:	Prof. XXXX
Main Advisor:	Prof. XXXX
Co-advisor(s):	Prof. XXXX (if any)
Internal TEC member 1:	Prof. XXXX
Internal TEC member 2:	Prof. XXXX
External TEC member 1:	Prof. XXXX (optional)
External TEC member 2:	Prof. XXXX (optional)

Declaration

I, Eric G. ROTHSTEIN MORRIS, declare that this thesis titled, "Latent-Behaviour Analysis" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

I hereby confirm the following:

- I hereby confirm that the thesis work is original and has not been submitted to any other University or Institution for higher degree purposes.
- I hereby grant SUTD the permission to reproduce and distribute publicly paper and electronic copies of this thesis document in whole or in part in any medium now known or hereafter created in accordance with Policy on Intellectual Property, clause 4.2.2.
- I have fulfilled all requirements as prescribed by the University and provided 1 copy of my thesis in PDF.
- I have attached all publications and award list related to the thesis (e.g. journal, conference report and patent).
- The thesis does / does not (delete accordingly) contain patentable or confidential information.
- I certify that the thesis has been checked for plagiarism via turnitin/ithenticate. The score is 100%.

Name and signature:

_____

Date:

_____

Abstract

Information Systems Technology and Design (ISTD)

Doctor of Philosophy

Latent-Behaviour Analysis

by Eric G. ROTHSTEIN MORRIS

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too...

Publications

Journal Papers, Conference Presentations, etc...

Acknowledgements

Pursuing a PhD has been the most interesting adventure I've ever had. Like any interesting adventure, it has had many types of moments: emotive, dramatic, suspenseful, disheartening, and fun. I am sure this story would have not been the same had I not had the support and patience from my friends, family and advisors.

I would first like to thank my family.

I will continue to write this once I am a bit less emotional.

Next, I would like to thank my supervisors Martín Ochoa, Sun Jun, and Sudipta Chattopadhyay. I've had the pleasure to work with each of you, learn from you, and experience through you the world of academia. I thank you for the time and lessons you've given me to make this journey a success.

I want to thank my friends, which I am incredibly fortunate to have in several parts of the world. Their constant support acted as a tailwind, pushing me forward closer to this achievement.

I will list all of them, indeed. They are important to me.

Contents

PhD Thesis Examination Committee i

Declaration ii

Abstract iii

Publications iv

Acknowledgements v

1 Preliminaries and Notation 1
- 1.1 Notation and Constructions on Sets and Functions . . . 1
  - 1.1.1 Sets and elements . . . 1
  - 1.1.2 Functions . . . 1
  - 1.1.3 Range set . . . 1
  - 1.1.4 Subsets $X \to 2$ . . . 2
  - 1.1.5 Exponential Set Y^X . . . 2
  - 1.1.6 Sum Set $X + Y$ . . . 2
  - 1.1.7 Product Set $X \times Y$ . . . 2
  - 1.1.8 Vector/Tuple $\vec{x}$ . . . 2
  - 1.1.9 Coordinates $\vec{x}[\pi]$ . . . 2
  - 1.1.10 Constant Function Δ_x . . . 2
  - 1.1.11 Pair Function (f, g) . . . 3
  - 1.1.12 Product Function $f \times g$ . . . 3
  - 1.1.13 Function Mapping $f(X)$ . . . 3
- 1.2 Category Theory Preliminaries and Notation . . . 3
  - 1.2.1 Categories, Objects and Arrows . . . 3
  - 1.2.2 Commutative Diagram . . . 3
  - 1.2.3 Initial and Terminal Objects . . . 4
  - 1.2.4 Functor . . . 4
- 1.3 The Category of Sets and Functions . . . 4
  - 1.3.1 Currying . . . 4
  - 1.3.2 Partial Application . . . 4
  - 1.3.3 Monoids and Sequences . . . 4
  - 1.3.4 Languages . . . 5
  - 1.3.5 Deterministic Finite Automata (DFA) . . . 5
  - 1.3.6 Set Isomorphisms . . . 5
    - List of Relevant Isomorphisms . . . 5
- 1.4 Universal (Co)Algebra Preliminaries and Notation . . . 6

1.4.1 F-(co)algebras . . . 6
1.4.2 Pointed coalgebras . . . 6
1.4.3 Automata as Pointed Coalgebras . . . 6
1.4.4 The Category of F-coalgebras and F-homomorphisms . . . 6
1.4.5 Semantic Map and Final F-Coalgebras . . . 6
1.4.6 Bisimilarity . . . 7
1.4.7 Coalgebraic Specification Language . . . 7
1.4.8 Completeness . . . 7
1.4.9 Soundness . . . 8
1.5 Coinduction . . . 8
1.5.1 Coinduction Proof Principle . . . 8
1.5.2 Bisimulation . . . 8
1.6 Cyber-Physical System . . . 8
1.7 Side Channels . . . 8

2 Foundations of Latent-Behaviour Analysis **9**
2.1 Introduction . . . 9
2.2 Motivational Example . . . 10
2.3 Latent F-coalgebras and Latent Behaviours . . . 13
2.4 Latent-Behaviour Analysis . . . 15
2.5 Some Applications of Latent-Behaviour Analysis . . . 16
2.5.1 Quantifying and Improving Robustness . . . 16
2.5.2 Classification of Attackers . . . 19
2.5.3 Side-Channel Repair . . . 20

A Appendix Title Here **22**

Bibliography **23**

List of Figures

1.1 Commutative diagram describing the equality $(h \circ g) \circ f = h \circ g \circ f = h \circ (g \circ f)$ 3

2.1 "The Arrow of Latent-Behaviour Analysis." This commutative diagram summarises the effect of spatial and dynamics transformations over the F-coalgebra (X, c), respectively modelled by m and b: the semantic map changes from $!_c$ to $!_{b \circ c \circ m}$. Through this work, we assume $b = \mathtt{id}$. 10
2.2 An automaton which recognises the language $(0|1)^*11$. 11
2.3 Automaton that models the transformed automaton, which now recognises $\varepsilon|(0|1)^*10$. 12
2.4 Composition of the fault m and the original behaviour for the states $(0, 0)$ and $(1, 1)$. The dotted, gray, bidirectional arrow in the centre models the effect of the fault m. The original behaviours appear as dashed, grey lines. The behaviour which results from the composition appears as solid, red lines. 13
2.5 Every F-coalgebra $(\sigma F, c)$ can be revealed from the final coalgebra $(\sigma F, 1_F)$ by means of a transformation m or a transformation b. In other words, it suffices to use spatial transformations m and the final F-coalgebra to reveal all F-coalgebras of σF, so dynamics transformations b are unnecessary. 15

List of Tables

For/Dedicated to/To my…

Chapter 1

Preliminaries and Notation

This section has content which I did not develop, but that I use. The content I developed follows in the next chapters.

1.1 Notation and Constructions on Sets and Functions

1.1.1 Sets and elements

We denote *sets* by upper-case letters X, Y, Z, etc, and we denote their *elements* by lower-case letters x, y, z, etc.

We denote the set of natural numbers by $\mathbb{N}$ and the set of natural numbers without zero by $\mathbb{N}^+$. Similarly, we denote the set of real numbers by $\mathbb{R}$ and the set of positive real numbers (excluding zero) by $\mathbb{R}^+$.

We denote the size of a finite set X by $|X|$.

1.1.2 Functions

We denote *functions* by lower-case letters f, g, h, etc. A function $f\colon X \to Y$ maps the elements from the set X to elements of the set Y. An *endofunction* is a function $g\colon X \to X$ which maps a set X to itself. We denote function composition by $\circ$. We say that an endofunction $h\colon X \to X$ has finite support iff $h(x) \neq x$ for a finite number of x in X, even if X is infinite.

1.1.3 Range set

A *range set* n where $n \in \mathbb{N}^+$ is a set of n elements, i.e., $n = \{0, 1, \ldots, n-1\}$. It is usually clear from the context when n represents a number or a range set. We highlight three important range sets: 0, 1 and 2.

The range set 0 is equal to the empty set $\emptyset$. The range set 1, which is equal to the set $\{0\}$, serves as a constant for several set operations (modulo isomorphism). The range set 2, which is equal to $\{0, 1\}$, to model booleans: 0 models `false` and 1 models `true`.

1.1.4 Subsets $X \to 2$

Given a set X, we characterise the *subsets of* X using functions of type $X \to 2$; we say that $x \in X$ is an element of the subset characterised by $f\colon X \to 2$ iff $f(x) = 1$.

1.1.5 Exponential Set Y^X

The *exponential set* Y^X is the set of all functions of type $X \to Y$. The exponential set $1 \to X$ is isomorphic to X, and we use it to select objects from X; more precisely, we choose an element $x \in X$ via a function $f\colon 1 \to X$ such that $f(0) = x$. (This seems overcomplicated at first, but we use this equivalence when discussing the following concepts of F-algebra and F-coalgebra.)

1.1.6 Sum Set $X + Y$

The *sum set* $X + Y$ is the disjoint union of X and Y, i.e.

$$X + Y \triangleq \{\iota_1(x) | x \in X\} \cup \{\iota_2(y) | y \in Y\}\,,$$

where ι_1 and ι_2 act as different tags.

1.1.7 Product Set $X \times Y$

The *product set* $X \times Y$ is the cartesian product of X and Y, i.e.

$$X \times Y \triangleq \{(x, y) | x \in X, y \in Y\}\,.$$

The *finite product set* of sets $X_1, \ldots, X_n$ is their cartesian product; i.e., $X_1 \times \ldots \times X_n$. We denote product sets by upper-case letters with an arrow above $\vec{X}, \vec{Y}, \vec{Z}$, etc.

1.1.8 Vector/Tuple $\vec{x}$

Given a finite product set $\vec{X} = X_1 \times \ldots \times X_n$, we denote the elements of $\vec{X}$ by $\vec{x}$, $\vec{y}$, $\vec{z}$, etc. We call these elements *vectors* or, alternatively, *tuples*.

1.1.9 Coordinates $\vec{x}[\pi]$

Given a finite product set $\vec{X} = X_1 \times \ldots \times X_n$, its *coordinates* are $\pi_1, \ldots, \pi_n$, where $\pi_i\colon \vec{X} \to X_i$ extract the i-th value of a vector; i.e., $\pi_i(x_1, \ldots, x_n) \triangleq x_i$, for $1 \leq i \leq n$. We often write the expression $\vec{x}[\pi]$ instead of $\pi(\vec{x})$ when π is a coordinate.

1.1.10 Constant Function Δ_x

Given sets X and Y, each element $x \in X$ lifts to a *constant function* $\Delta_x\colon Y \to X$, defined by $\Delta_x(y) = x$, for $y \in Y$.

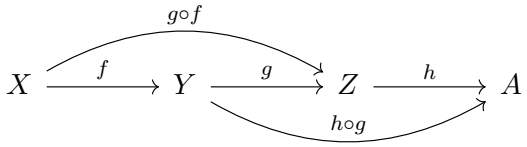


FIGURE 1.1: Commutative diagram describing the equality $(h \circ g) \circ f = h \circ g \circ f = h \circ (g \circ f)$

1.1.11 Pair Function (f, g)

Given two functions $f\colon X \to Y$ and $g\colon X \to Z$, the *pair function* $(f, g)\colon X \to Y \times Z$ is defined by $(f, g)(x) = (f(x), g(x))$.

1.1.12 Product Function $f \times g$

Given two functions $f\colon X \to Y$ and $g\colon A \to B$, the *product function* $f \times g\colon X \times A \to Y \times B$ is defined by $f \times g(x, a) = (f(x), g(a))$.

1.1.13 Function Mapping $f(X)$

Given a set X and a function $f\colon X \to Y$, the *mapping of f over X*, denoted $f(X)$, is the set defined by $f(X) \triangleq \{f(x) | x \in X\}$.

1.2 Category Theory Preliminaries and Notation

1.2.1 Categories, Objects and Arrows

A *category* $\mathbf{C}$ is a mathematical concept similar to a graph, comprised of two aspects: a collection of *objects*, denoted $Obj(\mathbf{C})$, and a collection of *arrows* among those objects, denoted $Arr(\mathbf{C})$. Each category satisfies the following rules:

- every object X has an identity arrow that maps X to itself, denoted $\mathtt{id}_X$,
- given the objects X, Y and Z, and the arrows $X \xrightarrow{f} Y$ and $Y \xrightarrow{g} Z$, the arrow $X \xrightarrow{g \circ f} Z$ exists; i.e. the composition of arrows is well defined,
- the identity arrows act as units for arrow composition; i.e., for $X \xrightarrow{f} Y$, the equality $\mathtt{id}_Y \circ f = f = f \circ \mathtt{id}_X$ holds.
- the composition of arrows is associative; i.e., $(h \circ g) \circ f = h \circ (g \circ f)$.

1.2.2 Commutative Diagram

A *commutative diagram* is a directed graph which visually represents equations. Like any directed graph, source nodes do not have incoming arrows, and sink nodes do not have outgoing arrows, and they represent the beginning and end of equations. For example, given the functions $f\colon X \to Y$, $g\colon Y \to Z$ and $h\colon Z \to A$, the equalities $(h \circ g) \circ f = h \circ g \circ f = h \circ (g \circ f)$ is represented by the diagram shown in Figure 1.1.

1.2.3 Initial and Terminal Objects

A category $\mathbf{C}$ has an *initial object*, often denoted by 0, if there exists a unique arrow from 0 to every object o in the category. Dually, a category $\mathbf{C}$ has a *final object*, often denoted by 1, if there exists a unique arrow from every object o in the category to 1.

A category may have several different initial and terminal objects, but they are always unique modulo isomorphism.

1.2.4 Functor

Functors are to categories what arrows are to objects. A *functor* F maps a category $\mathbf{C}$ to a category $\mathbf{D}$, mapping $Obj(\mathbf{C})$ to $Obj(\mathbf{D})$, and $Arr(\mathbf{C})$ to $Arr(\mathbf{D})$ such that the following conditions are satisfied for all objects X, Y, Z and arrows f, g.

- for $f\colon X \to Y$, f is mapped to a function of type $g\colon F(X) \to F(Y)$,
- $\mathtt{id}_X$ is mapped to $\mathtt{id}_{F(X)}$; i.e. $F(\mathtt{id}_X) = \mathtt{id}_{F(X)}$.
- if $f\colon X \to Y$ and $g\colon Y \to Z$, then $F(g \circ f) = F(g) \circ F(f)$.

These conditions ensure functors preserve the structure of the category they are mapping.

1.3 The Category of Sets and Functions

The *category of sets and functions* $\mathbf{Set}$ is the category whose objects are sets and whose arrows are functions. The empty set 0 is the initial object, and any set isomorphic to the range set 1 is a terminal object.

1.3.1 Currying

Currying is the use of the isomorphism between the sets $Z^{X \times Y}$ and Z^{Y^X} which maps a function $f\colon (X \times Y) \to Z$ to a function $g\colon X \to Z^Y$ such that $g(x)(y) = f(x, y)$ for $x \in X$ and $y \in Y$. We use currying to adjust the types between equivalent constructions (e.g. between Automata and F-coalgebras).

1.3.2 Partial Application

Given a function $f\colon X \to Y \to Z$ and $x \in X$, the *partial application* of f given x is the function $f_x\colon Y \to Z$, defined for $y \in Y$ by $f_x(y) = f(x)(y)$.

1.3.3 Monoids and Sequences

A *monoid* in the category $\mathbf{Set}$ is a set X paired with a sum function $+\colon X \to X \to X$ and a unit element $0 \in X$ such that 0 is neutral for $+$ and $+$ is associative.

We highlight two monoids for a given set X: the *free monoid of* X and the *endofunctions monoid of* X.

The *free monoid of* X is the set X^* of finite sequences of elements of X, where the sum function is sequence concatenation $\cdot$ and the unit is the empty sequence ε.

The *endofunctions monoid of* X is the set X^X of endofunctions in X, where the sum is function composition $\circ$ and the unit is the identity function $\mathtt{id}_X$.

1.3.4 Languages

Given a set X, a *language with alphabet* X is a subset of X^*.

1.3.5 Deterministic Finite Automata (DFA)

Given a finite set A, a *deterministic finite automaton* (DFA) is a tuple $\mathbb{X} = (X, x_0, \delta, F)$, where X is a finite set, x_0 is an element of X, $\delta\colon X \times A \to X$ is a transition function, and $F\colon X \to 2$ is a subset of X which marks accepting states. Given a sequence $w \in A^*$, we say that the automaton $\mathbb{X}$ *accepts* w if and only if the following conditions are satisfied,

- if $w = \varepsilon$, then $\mathbb{X}$ accepts w iff $F(x_0) = 1$,
- if $w = a \cdot w'$, then $\mathbb{X}$ accepts w iff the automaton $(X, \delta(x_0, a), \delta, F)$ accepts w', with $a \in A$ and $w' \in A^*$.

The denotational semantics of a DFA is the language with alphabet A whose sequences it accepts, rejecting the rest.

1.3.6 Set Isomorphisms

Homomorphisms and isomorphisms

List of Relevant Isomorphisms

- $1 \times X \simeq X$,
- $2 \times X \simeq X + X$,
- $X^1 \simeq X$,
- $X^2 \simeq X \times X$,
- $Z^{X \times Y} \simeq Z^{Y^X}$,
- $X \times Y \simeq Y \times X$
- $X + Y \simeq Y + X$
- $|X| = n$ iff $X \simeq n$.
- $\underbrace{(X \times \ldots \times X)}_{n \text{ times}} \simeq X^n$ (for this case, we say that n is the set of coordinates of X^n)

1.4 Universal (Co)Algebra Preliminaries and Notation

1.4.1 F-(co)algebras

Given an endofunctor F in a category $\mathbf{C}$, an *F-coalgebra* is a pair $(X, X \xrightarrow{c} F(X))$ of an object X and a morphism $c\colon X \to F(X)$. We use F-coalgebras to model dynamic systems with state. We call X the *carrier* and c the *dynamics*.

Dually, an *F-algebra* is a pair $(X, F(X) \xrightarrow{a} X)$.

1.4.2 Pointed coalgebras

Pointed coalgebras are coalgebras with a distinguished state, usually referred to as the *initial state*. Formally, for a functor F, a pointed F-coalgebra is a triple (X, c, x_0) where (X, c) is an F-coalgebra and (X, x_0) is a Δ_X-algebra, i.e., $x_0\colon 1 \to X$, and $x_0(\star)$ characterises the initial state.

1.4.3 Automata as Pointed Coalgebras

DFA which characterise languages with an alphabet A can be modelled by coalgebras of the functor $F(X) = 2 \times X^A$. Given a DFA (X, x_0, δ, F), the corresponding F-coalgebra is $(X, (F, \delta))$, where $(F, \delta)\colon X \to 2 \times X^2$ is a function pair. We can make this coalgebra pointed by adding a function $x_0\colon 1 \to X$ to mark the initial state.

In general, given sets I and O, we can model transition systems whose denotational semantics are functions of type $I^* \to O$; i.e., systems which process a finite sequence of elements in I and respond with an element in O. The corresponding functor in this case is $F(X) = O \times X^I$. An $F-$coalgebra would be of the form $(X, X \xrightarrow{(\gamma,\delta)} O \times X^I)$, with $\gamma\colon X \to O$ and $\delta\colon X \to X^I$; the function o lets us explore the component in the state x, and δ lets us perform transitions. We write x/o to denote $\gamma(x) = o$, and we write x^i as a shorthand for $\delta(x)(i)$.

1.4.4 The Category of F-coalgebras and F-homomorphisms

Given a functor F, the *category of F-coalgebras*, denoted $\mathbf{Coalg_F}$ is the category whose objects are F-coalgebras and whose arrows are *F-homomorphisms*. An *F-homomorphism* $\mathbb{X} \xrightarrow{f} \mathbb{Y}$ between an F-coalgebra $\mathbb{X} = (X, c)$ and an F-coalgebra $\mathbb{Y} = (Y, d)$ is a function $f\colon X \to Y$ such that

$$F(f) \circ c = d \circ f. \tag{1.1}$$

1.4.5 Semantic Map and Final F-Coalgebras

Let F be a functor such that $F(X) = O \times (X)^I$, and let I and O be sets; we define the set $\sigma F \triangleq O^{I^*}$, and we define the function pair $(?, (\cdot)')\colon \sigma F \to O \times (\sigma F)^I$, for $\phi \in \sigma F$, $i \in I$ and $w \in I^*$, by

$$\phi? \triangleq \phi(\varepsilon), \phi'(i)(w) \triangleq \phi(i \cdot w) \tag{1.2}$$

Following convention, we write $\phi'(i)$ as ϕ^i, and the pair $(?, (\cdot)')$ as 1_F. The pair $(\sigma F, 1_F)$ is a *final F-coalgebra*, since it is a final object in the category $\mathbf{Coalg_F}$.

Given an F-coalgebra (X, c) where $c = (\gamma, \delta)$, the unique F-homomorphism between (X, c) and $(\sigma F, 1_F)$ is given by a function $!_c \colon X \to \sigma F$, which is defined for $x \in X$, $i \in I$ and $w \in I^*$ by

$$!_c(x)(\varepsilon) \triangleq \gamma(x), \quad \text{and} \quad !_c(x)(i \cdot w) \triangleq !_c(x^i)(w). \tag{1.3}$$

The function $!_c$ is called the *semantic map* because it maps $x \in X$ to its denotational semantics in σF. For example, in the case of DFA as coalgebras, it maps each state x in the carrier of the coalgebra to the language it would be recognised if x were the initial state in the DFA.

1.4.6 Bisimilarity

Given F-coalgebras (X, c) and (Y, d), we say that states $x \in X$ *and* $y \in Y$ *are bisimilar*, denoted $x \sim y$, if and only if $!_c(x) = !_d(y)$. In other words, x and y are bisimilar iff they have the same semantics.

1.4.7 Coalgebraic Specification Language

A *coalgebraic specification language* for a functor $F(X) = O \times X^I$ is an F-coalgebra $(\mathcal{E}, c)$ where $\mathcal{E}$ is a set of expressions whose semantics are given by the semantic map $!_c$.

For example, consider the functor $F(X) = 2 \times X^2$; we define the set of expressions $\mathcal{E}$ by the grammar

$$e ::= ID : 0?ID \wedge 1?ID \wedge \downarrow (0|1) \tag{1.4}$$

where ID is a set of identifiers. We define the pair function $c = (\gamma, \delta) \colon \mathcal{E} \to F(\mathcal{E})$ by

$$\begin{aligned} \gamma(x : 0? \wedge 1?z \wedge \downarrow a) &\triangleq a \\ \delta(x : 0?y \wedge 1?z \wedge \downarrow a)(0) &\triangleq y \\ \delta(x : 0?y \wedge 1?z \wedge \downarrow a)(1) &\triangleq z. \end{aligned}$$

For example, the expression $x : 0?x \wedge 1?x \wedge \downarrow 1$ can be considered to describe a single-state automaton whose single state is accepting, and it loops to itself on both inputs 0 and 1. The semantic map $!_c$ maps the expression $x : 0?x \wedge 1?x \wedge \downarrow 1$ to the language 2^*.

Cite Kleene Coalgebra somehow

1.4.8 Completeness

Given a coalgebraic specification language $L = (\mathcal{E}, c)$ for the functor $F(X) = O \times X^I$, we say that L is *complete* if and only if the semantic map $!_c \colon \mathcal{E} \to \sigma F$ is surjective. In other words, for every behaviour $\phi \in \sigma F$, there exists an expression $e \in \mathcal{E}$ such that $!_c(e) = \phi$.

1.4.9 Soundness

Given a coalgebraic specification language $L = (\mathcal{E}, c)$ for the functor $F(X) = O \times X^I$ and a notion of equality in $\mathcal{E}$ modelled by a function $=: \mathcal{E} \times \mathcal{E} \to 2$, we say that L is *sound* if and only if, for $e_1, e_2 \in \mathcal{E}$, if $e_1 = e_2$, then $e_1 \sim e_2$; i.e., if two expressions are equal, then they have the same behaviour.

1.5 Coinduction

Coinduction is a dual principle to induction, and we use it to define what things are in terms of how they behave.

I'll come back to this

1.5.1 Coinduction Proof Principle

Coinductive Definitions?

1.5.2 Bisimulation

1.6 Cyber-Physical System

This section probably makes more sense in its own chapter, when we explain what the latent behaviour of a CPS is.

1.7 Side Channels

Chapter 2

Foundations of Latent-Behaviour Analysis

2.1 Introduction

A *latent behaviour* is a functionality that is not coded into a system, so the system normally does not display it, but given extraordinary circumstances, the system displays such functionality. To reveal a latent behaviour, we transform the state space of the system. Normally, the state space is not transformed, so we can model "extraordinary circumstances" using these transformations. This concept is probably well illustrated by *return-oriented programming* (ROP) Shacham, 2007, where the state of the call stack is transformed to alter the behaviour of the program being executed, but the program itself is not modified.

In ROP, the adversary manipulates the call stack to string code gadgets together and arbitrarily execute code. However, if the program is very small and does not have enough gadgets, then the computational power of the attacker is restricted, since it becomes impossible to create certain behaviours. This restriction is similar to the restrictions we face when working with non-Turing complete languages (e.g., we cannot use regular expressions to specify arbitrary Turing machines). However, given enough gadgets, we can reveal any behaviour we want if we appropriately compose the gadgets.

In this section, we explain the commutative diagram presented in Figure 2.1. This diagram, which we refer to as the "Arrow" of latent-behaviour analysis, illustrates how spatial and dynamics transformations affect the behaviour of an arbitrary F-coalgebra (X, c) by changing the semantic map from $!_c$ to $!_{b \circ c \circ m}$. The intuition behind the diagram of the Arrow is the following. Normally, an F-coalgebra (X, c) gives semantics to the elements in X via the semantic map $!_c$. When affected by a spatial transformation $m \colon X \to X$ and a dynamics transformation $b \colon F(X) \to F(X)$, the F-coalgebra (X, c) reveals an F-coalgebra $(X, b \circ c \circ m)$. The behaviour of elements in $(X, b \circ c \circ m)$ may bear no resemblance to the original behaviour. At most, we can say that if m and b preserve bisimilarity, then there exists a transformation $\omega \colon \sigma F \to \sigma F$ in the carrier of the final F-coalgebra $(\sigma F, 1)$ which maps the original behaviours of the system into the new behaviours revealed by m and b.

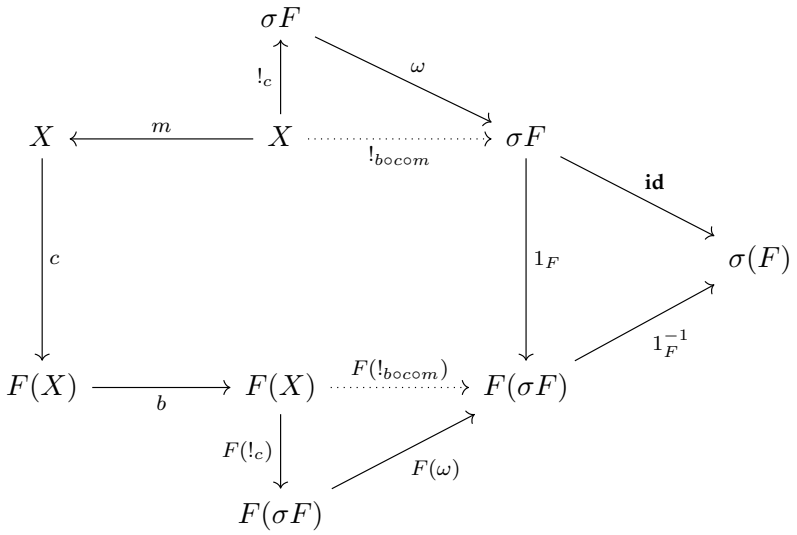


FIGURE 2.1: "The Arrow of Latent-Behaviour Analysis." This commutative diagram summarises the effect of spatial and dynamics transformations over the F-coalgebra (X, c), respectively modelled by m and b: the semantic map changes from $!_c$ to $!_{bocom}$. Through this work, we assume $b = \texttt{id}$.

2.2 Motivational Example

Consider the automaton shown in Figure 2.2, which recognises the language of sequences in 2^* that end in two consecutive 1; i.e., the language $(0|1)^*11$. This automaton is defined by the tuple $(\vec{X}, \vec{x}_0, \delta, F)$; the carrier is $\vec{X} \triangleq 2 \times 2$, the initial state selector is $\vec{x}_0 \colon 1 \to \vec{X}$ with $\vec{x}_0(0) \triangleq (0,0)$, the transition function is $\delta \colon \vec{X} \to 2 \to \vec{X}$, defined for $\vec{x} \in \vec{X}$ and $i \in 2$ by $\delta(\vec{x})(i) \triangleq (i, \vec{x}[0])$, and the characteristic predicate of the set of accepting states is $F \colon \vec{X} \to 2$, where $F(x, y) \triangleq x \wedge y$ (i.e. $(1,1)$ is the only accepting state). This automaton is not minimal, since the states $(0,0)$ and $(0,1)$ are different, yet they are bisimilar.

Now, consider a transformation $m \colon \vec{X} \to \vec{X}$, defined by $m(\vec{x}) \triangleq (\neg\vec{x}[0], \neg\vec{x}[1])$ for $\vec{x} \in \vec{X}$; i.e., m maps (a, b) to $(\neg a, \neg b)$. This transformation models some fault which could be introduced by a malicious programmer or by an attacker which corrupts the state of the system. Due to this transformation, if the current state is $\vec{x} = (a, b)$, when we intend to check whether $\vec{x}$ is accepting, we check whether $(\neg a, \neg b)$ is accepting instead, and when we intend to compute $\delta(\vec{x})(i)$, we instead compute $\delta(\neg a, \neg b)(i)$.

Figure 2.3 shows a model of the transformed automaton. This automaton recognises the language of sequences that are either empty or end in 10; i.e., $\varepsilon|(0|1)^*10$. Before we explain why the automaton in Figure 2.3 is in fact the resulting system when we apply the transformation m, we can first test if the transformed system recognises sequences $\varepsilon|(0|1)^*10$ due to the transformation m. To do so, let us consider pairs of states $[\vec{x}, \vec{y}]$ where $\vec{x} = (x_1, x_2)$ and $\vec{y} = (y_1, y_2)$ such that the pair $\vec{x}$ follows the original behaviour while $\vec{y}$ follows the new behaviour. Let $[(0,0),(0,0)]$ be the initial state, and let $\delta_i \colon \vec{X} \to$

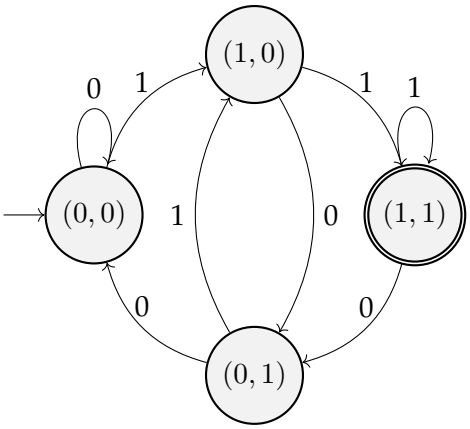


FIGURE 2.2: An automaton which recognises the language $(0|1)^*11$.

$\vec{X}$ be the functions defined by $\delta_i(\vec{x}) = \delta(\vec{x})(i)$ for $i \in 2$ (i.e., δ_0 is a partial application of δ given 0 and δ_1 is a partial application of δ given 1); the trace of the sequence 00 is

$$\begin{aligned}[(0,0),(0,0)] &\xrightarrow{\mathtt{id}\times m} [(0,0),(1,1)] \xrightarrow{\delta_0\times\delta_0} [(0,0),(0,1)] \\ &\xrightarrow{\mathtt{id}\times m} [(0,0),(1,0)] \xrightarrow{\delta_0\times\delta_0} [(0,0),(0,1)] \\ &\xrightarrow{\mathtt{id}\times m} [(0,0),(1,0)] \xrightarrow{F\times F} [0,0],\end{aligned}$$

so 00 is rejected by both automata. Now, if we receive the sequence 10, the resulting state trace is

$$\begin{aligned}[(0,0),(0,0)] &\xrightarrow{\mathtt{id}\times m} [(0,0),(1,1)] \xrightarrow{\delta_1\times\delta_1} [(1,0),(1,1)] \\ &\xrightarrow{\mathtt{id}\times m} [(1,0),(0,0)] \xrightarrow{\delta_0\times\delta_0} [(0,1),(0,0)] \\ &\xrightarrow{\mathtt{id}\times m} [(1,0),(1,1)] \xrightarrow{F\times F} [0,1];\end{aligned}$$

the transformed automaton accepts 10, but the original automaton does not. The trace of the sequence 11 is

$$\begin{aligned}[(0,0),(0,0)] &\xrightarrow{\mathtt{id}\times m} [(0,0),(1,1)] \xrightarrow{\delta_1\times\delta_1} [(1,0),(1,1)] \\ &\xrightarrow{\mathtt{id}\times m} [(1,0),(0,0)] \xrightarrow{\delta_1\times\delta_1} [(1,1),(1,0)] \\ &\xrightarrow{\mathtt{id}\times m} [(1,1),(0,1)] \xrightarrow{F\times F} [1,0],\end{aligned}$$

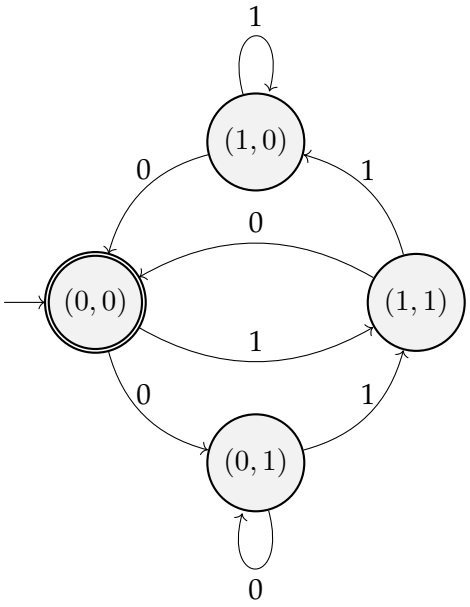


FIGURE 2.3: Automaton that models the transformed automaton, which now recognises $\varepsilon|(0|1)^*10$.

so 11 is accepted by the original automaton, but rejected by the transformed automaton. For the sequence 110, the resulting state trace is

$$\begin{aligned}[(0,0),(0,0)] &\xrightarrow{\mathtt{id}\times m} [(0,0),(1,1)] \xrightarrow{\delta_1\times\delta_1} [(1,0),(1,1)] \\ &\xrightarrow{\mathtt{id}\times m} [(1,0),(0,0)] \xrightarrow{\delta_1\times\delta_1} [(1,1),(1,0)] \\ &\xrightarrow{\mathtt{id}\times m} [(1,0),(0,1)] \xrightarrow{\delta_0\times\delta_0} [(0,1),(0,0)] \\ &\xrightarrow{\mathtt{id}\times m} [(0,1),(1,1)] \xrightarrow{F\times F} [0,1];\end{aligned}$$

the transformed automaton accepts 110, but the original automaton does not. Finally, the original automaton rejects the empty sequence ε, but the transformed automaton accepts it, since

$$[(0,0),(0,0)] \xrightarrow{\mathtt{id}\times m} [(0,0),(1,1)] \xrightarrow{F\times F} [0,1].$$

We obtain the automaton shown in Figure 2.3 by "copying" the original behaviour from images of m to their preimages. Figure 2.4 shows this procedure for $(0,0)$ and $(1,1)$. This includes copying the behaviour under δ and under F.

We preserve the initial state because $\vec{x}_0$ is a selection/construction/algebraic operation, not a dynamics/behavioural/coalgebraic operation. We do not apply transformations to the results of algebraic operations to avoid aggregating the transformation erroneously due to composition of algebraic and coalgebraic operations. Consider the followign: the initial state $\vec{x}_0$ is of type $1 \to \vec{X}$ which is algebraic, so the only way to apply m to $\vec{x}_0$ is by composing it on the left, i.e., $m \circ \vec{x}_0(0)$. Checking if the initial state is

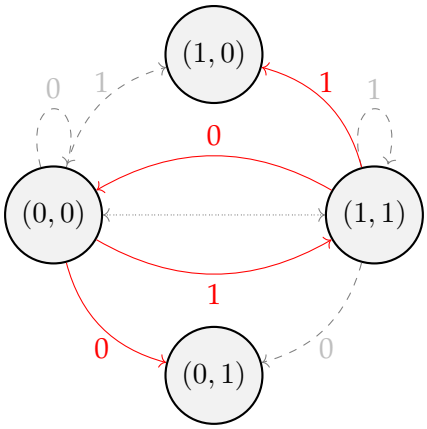


FIGURE 2.4: Composition of the fault m and the original behaviour for the states $(0, 0)$ and $(1, 1)$. The dotted, gray, bidirectional arrow in the centre models the effect of the fault m. The original behaviours appear as dashed, grey lines. The behaviour which results from the composition appears as solid, red lines.

accepting in the original automaton corresponds to the expression $(F \circ \vec{x}_0)(*)$; however, the expression

$$(F \circ m) \circ (m \circ \vec{x}_0)(0) = (F \circ m)(1, 1) = F(0, 0) = 0,$$

applies m twice, and it fails to properly check if the initial state of the transformed automaton is final. Instead, the correct expression is

$$(F \circ m)(\vec{x}_0) = (F \circ m)(0, 0) = F(1, 1) = 1.$$

We say that the transformed system is *latent* with respect to the original system, since the application of the transformation m reveals the new behaviour. In general, we cannot reveal arbitrary behaviours only through transformations of the state space; e.g., we cannot obtain an automaton that accepts every sequence from a single-state automaton that rejects all sequences by just transforming the state space.

In the following, we present a general treatment for spatial transformations and latent behaviours in the context of F-coalgebras. We describe the limitations on revealing new behaviours through spatial transformations in arbitrary F-coalgebras, and we also describe the necessary conditions over that lift these limitations.

2.3 Latent F-coalgebras and Latent Behaviours

Given an F-coalgebra $(X, c\colon X \to F(X))$, each state $x \in X$ has an associated dynamics function $c(x) \in F(X)$. In this section, we study the effect of a state transformation $m\colon X \to X$ over the F-coalgebra (X, c), which changes the dynamics of $x \in X$ from $c(x)$ to $c \circ m(x)$.

In terms of types, we can always compose the function c with any carrier transformation m; the composition $c \circ m \colon X \to F(X)$ exists and is well defined. However, the behaviour of elements might be greatly affected. In particular, states which are bisimilar under c might no longer be bisimilar under $c \circ m$. If m preserves bisimilarity, then we say that m is *behaviourally consistent*.

Definition 2.3.1 (Behaviourally Consistent Spatial Transformations)**.** Given an F-coalgebra (X, c), any function of type $m \colon X \to X$ is a *spatial transformation*. A spatial transformation m is *behaviourally consistent* if and only if, whenever $x \sim_c y$, then $m(x) \sim_c m(y)$, for all $x, y \in X$.

Corollary 2.1. If (X, c) is a minimal F-coalgebra, then every spatial transformation is behaviourally consistent. This property also holds if (X, c) is a sound coalgebraic specification language.

Applying transformation function $m \colon X \to X$ in the context of an F-coalgebra (X, c) changes the normal behaviour of the states in X, and it reveals the *latent coalgebra of* (X, c) *under* m.

Definition 2.3.2 (Latent Coalgebra and Latent Behaviour)**.** Given an F-coalgebra (X, c) and a transformation m, the *latent coalgebra of* (X, c) *under* m is $(X, c \circ m)$.

The semantic map $[\![x]\!]_{com}$ of the latent coalgebra $(X, c \circ m)$ characterises the *latent behaviours* revealed m.

Example 2.1. Consider the automaton from Section 2.2 which recognises the language $(0|1)^*11$. Let F be the functor $F(X) = 2 \times X^2$; we model the automaton with an F-coalgebra $(\vec{X}, (F, \delta))$, where $\vec{X} = 2 \times 2$ and $(F, \delta) \colon X \to 2 \times X^2$ is a function pair, defined for $(x, y) \in \vec{X}$ by

$$F(x, y) \triangleq x \wedge y \tag{2.1}$$

$$\delta(x, y)(i) \triangleq (i, x). \tag{2.2}$$

Just like its automaton counterpart, this F-coalgebra is not minimal, since $(0, 0)$ and $(0, 1)$ are bisimilar There are $|X|^{|X|} = 256$ different spatial transformations, but that does not imply the existence of 256 different latent coalgebras; e.g., the transformations $\Delta_{(0,0)}$ and $\Delta_{(0,1)}$ which respectively map all states to $(0, 0)$ and $(0, 1)$ yield isomorphic latent coalgebras, since $(0, 0)$ and $(0, 1)$ are bisimilar; both of these latent coalgebras recognise the empty language starting in all states. The transformation $m(x, y) = (\neg x, \neg y)$ reveals the latent coalgebra where

$$(F \circ m)(x, y) = \neg x \wedge \neg y \tag{2.3}$$

$$(\delta \circ m)(x, y)(i) = (i, \neg x). \tag{2.4}$$

The behaviour of a state x changes when m acts on x. In particular, the image of state $(0, 0)$ under the semantic map is no longer the language $(0|1)^*11$; it is the language $\varepsilon|(0|1)^*10$ instead.

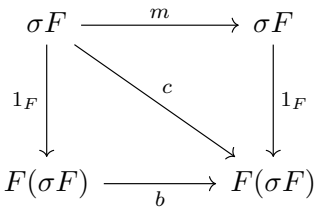


FIGURE 2.5: Every F-coalgebra $(\sigma F, c)$ can be revealed from the final coalgebra $(\sigma F, 1_F)$ by means of a transformation m or a transformation b. In other words, it suffices to use spatial transformations m and the final F-coalgebra to reveal all F-coalgebras of σF, so dynamics transformations b are unnecessary.

2.4 Latent-Behaviour Analysis

LBA is the study of the effects that a spatial transformation $m\colon X \to X$ has over the behaviour of states in X in the context of an F-coalgebra $(X, X \xrightarrow{c} F(X))$. The choice to focus on spatial transformations seems arbitrary; clearly, it is also possible to affect the behaviour of states by composing c with a dynamics transformation $b\colon F(X) \to F(X)$ to obtain the new dynamics function $b \circ c$, as shown in the "Arrow" from Figure 2.1. However, there are systems for which spatial and behavioural transformations can reveal the same behaviours; moreover, we can reveal any arbitrary behaviours via spatial transformations: *final F-coalgebras*, and *sound and complete coalgebraic specification languages*.

A final F-coalgebra $(\sigma F, 1_F)$ has a dynamics function $1_F\colon \sigma F \to F(\sigma F)$ that is an isomorphism. Since $\sigma F \simeq F(\sigma F)$, it naturally follows that

$$\sigma F \to \sigma F \simeq \sigma F \to F(\sigma F) \simeq F(\sigma F) \to F(\sigma F). \tag{2.5}$$

In other words, in the carrier of the final F-coalgebra, spatial transformations (of type $\sigma F \to \sigma F$), dynamics transformations (of type $F(\sigma F) \to F(\sigma F)$) and F-coalgebras (of type $\sigma F \to F(\sigma F)$) are in a one-to-one correspondence.

Intuition tells us that $1_F\colon \sigma F \to F(\sigma F)$ should correspond to $\mathtt{id}_{\sigma F}\colon \sigma F \to \sigma F$ and should correspond to $\mathtt{id}_{F(\sigma F)}\colon F(\sigma F) \to F(\sigma F)$. The general correspondence is given by the following proposition.

Proposition 2.4.1. For every F-coalgebra $(\sigma F, c)$, there are transformations $m\colon \sigma F \to \sigma F$ and $b\colon F(\sigma F) \to F(\sigma F)$ such that the diagram in Figure 2.5 commutes.

Proof. Since $1_F\colon \sigma F \to F(\sigma F)$ is an isomorphism, by taking $m = 1_F^{-1} \circ c$ and $b = c \circ 1_F^{-1}$, the diagram in Figure 2.5 commutes. □

Corollary 2.2. Every F-coalgebra $(\sigma F, c)$ is a latent coalgebra of $(\sigma F, 1_F)$ under some spatial transformation $m\colon \sigma F \to \sigma F$.

This property applies to the carrier of the final coalgebra because of the reversibility of the final dynamics map 1_F, which formalises a correspondence between state and

behaviour. This property is also shared by sound and complete coalgebraic specification languages, since their semantic map is an epimorphism. For an arbitrary F-coalgebra (X, c), only some latent F-coalgebras can be revealed by spatial transformations $m\colon X \to X$. It would not be an exaggeration to state that F-coalgebras which satisfy this property have all the "necessary gadgets" (with respect to the functor F) to reveal any arbitrary functionality using spatial transformations.

Given a final F-coalgebra $(\sigma F, 1_F)$ and an arbitrary coalgebra $(\sigma F, c)$, the spatial transformation $\omega\colon \sigma F \to \sigma F$, defined by $\omega \triangleq 1_F^{-1} \circ c$, implements c. The identity spatial transformation implements 1_F. This duality between F-coalgebras and spatial transformations in σF enables a natural composition of F-coalgebras in σF.

Given two F-coalgebras $(\sigma F, c_1)$ and $(\sigma F, c_2)$ whose respective spatial transformation implementations are ω_1 and ω_2, the composition $\omega_2 \circ \omega_1$ gives rise to the following equivalent concepts:

- the F-coalgebra $(\sigma F, c_2 \circ 1_F^{-1} \circ \omega_1)$,
- the latent coalgebra of $(\sigma F, c_2)$ under ω_1,
- the latent coalgebra of $(\sigma F, 1_F)$ under $\omega_2 \circ \omega_1$.

In summary, we can represent F-coalgebras whose carrier is σF by endofunctions in σF, and we can reason about their composition as we would with functions in the monoid of endofunctions $(\sigma F \to \sigma F, \circ, \mathtt{id})$.

In the following, we return to arbitrary F-coalgebras (X, c), but we keep in mind the following: when we reveal a latent coalgebra of (X, c) under a spatial transformation m, we are implicitly sequentially composing the behaviour function c with the behaviour function of the F-coalgebra implemented by m; first we apply m and then we apply c.

2.5 Some Applications of Latent-Behaviour Analysis

We study the effects that spatial transformations have on the behaviour of the system. Just like in geometry, some spatial transformations may preserve some properties of the system, while others may break them. In this section, we present three ideas for LBA to study three aspects of systems: *quantifying and improving robustness*, *classification of attackers*, and *side-channel repair*.

2.5.1 Quantifying and Improving Robustness

Informally, the robustness of a system quantifies how much stress the system can withstand without compromising its functionality.

Find some nice definitions of robustness.

In this section, we explore the idea of modelling stress via spatial transformations, and how we can quantify and improve the robustness of a system by studying properties of latent coalgebras.

To formally capture the notion of whether the functionality of a system is compromised, we use *behavioural properties*.

Definition 2.5.1 (Behavioural Property)**.** Let F be a functor for which a final F-coalgebra $(\sigma F, 1)$ exists. A *behavioural property* P is a function $P\colon \sigma F \to 2$. Given an arbitrary F-coalgebra (X, c), we say that $x \in X$ satisfies the behavioural property P if and only if $(P \circ !_c)(x) = 1$.

Behavioural properties come in all shapes and forms, and are often described using logics like LTL **LTL**. CTL **CTL**, and μ calculus **MuCalculus**. Let us for now assume that we have a set of behavioural properties $\mathcal{R} = \{R_1, \ldots, R_n\}$ which model both functional and security requirements of the system. We discuss concrete ways to define these behavioural properties in Chapters **??** and **??**.

Consider the following problem: we are given an arbitrary F-coalgebra (X, c), an initial state $x_0 \in X$, and a set of behavioural properties $\mathcal{R} = \{R_1, \ldots, R_n\}$. Normally, we would check if x_0 satisfies all behavioural properties in $\mathcal{R}$, but now we consider a spatial transformation $m\colon X \to X$, and we want to check if x_0 still satisfies all the given behavioural properties in the resulting latent coalgebra revealed by m.

We might think that it suffices to check $(R_i \circ !_c)(m(x_0)) = 1$ for each requirement R_i, but that would be incorrect as illustrated by the following example.

Example 2.2 (Wrong Verification)**.** In the context of Example 2.1, the functor is $F(X) = 2 \times X^2$, and it has a final F-coalgebra $(2^{2^*}, (\varepsilon?, (\cdot)'))$ where $\varepsilon?\colon 2^{2^*} \to 2$ checks whether the empty sequence ε belongs to the language and $(\cdot)'\colon 2^{2^*} \to (2 \to 2^{2^*})$ computes the Brzozowski derivative **BrzozowskiDerivative** of the language.

Let $R\colon 2^{2^*} \to 2$ be the behavioural property defined by $R(\phi) \triangleq \phi \sim \varepsilon|(0|1)^*10$, for $\phi \in 2^{2^*}$. The property R is not satisfied by any state of the original F-coalgebra, so $(R \circ !_c)(m(x_0))$ would be 0 for all spatial transformations m; nevertheless, R is satisfied by x_0 in the latent coalgebra revealed by the spatial transformation $m(x, y) = (\neg x, \neg y)$ used in Example 2.1, since it is the language the state $(0, 0)$ recognises in the latent coalgebra.

Part of the novelty of LBA is that we can study a latent coalgebra $(X, c \circ m)$ just as we would study the original coalgebra (X, c). By this, we mean that we can apply testing and verification techniques without complication. In the following, we illustrate how we can use spatial transformations to model attacks which target the state of systems rather than the program, and we study the details of this approach in Chapter **??** when we apply it to cyber-physical systems.

Definition 2.5.2 (State/Behaviour-based Attacks)**.** Given an F-coalgebra (X, c), we define the set of its *state-based attacks* corresponds to be the set of its spatial transformations, i.e., X^X. We also define the set of its *behaviour-based attacks* by $F(X)^{F(X)}$.

Under this definition, LBA studies the effect of state-based attacks. Henceforth, when we refer to an attack, we implicitly mean a state-based attack (unless stated otherwise). In the particular case of final F-coalgebras, their state-based attacks coincide with their behaviour-based attacks.

We know that an attack m reveals a latent coalgebra $(X, c \circ m)$ which may or may not satisfy the same behavioural properties that (X, c) satisfies. We could check whether $(X, c \circ m)$ satisfies all requirements in $\mathcal{R}$, but this is not very informative if we arbitrarily choose m. We propose to systematically generate a set of spatial transformations $\mathcal{M} = \{m_1, \ldots, m_p\}$ based on some attacker model, and test whether each revealed latent coalgebra $(X, c \circ m_i)$ satisfies each requirement R_j for $m_i \in \mathcal{M}$ and $R_j \in \mathcal{R}$. If a requirement R is not satisfied in some latent coalgebra $(X, c \circ m)$, then the attacker can use m to break R in the original system (X, c); in this case, we say that *the attack m breaks the requirement R*. If no attack in $\mathcal{M}$ breaks any requirement in $\mathcal{R}$, then we say that the system is *robust* against the attacker model that generated $\mathcal{M}$. This approach gives us a qualitative notion of robustness.

We can extend qualitative robustness into quantitative robustness by discounting broken requirements. More precisely, given (X, c), $\mathcal{M}$, and $\mathcal{R}$ such that $|\mathcal{R}| = n$, we can define the set $\mathcal{M}[\mathcal{R}]$ of requirements in $\mathcal{R}$ that are broken by one or more attacks in $\mathcal{M}$; similarly $m[\mathcal{R}]$ is the set of requirements that m breaks. We estimate the robustness of the system by the formula

$$\sum_{i=i}^{n} w_i(1 - (R_i \in \mathcal{M}[\mathcal{R}])),$$

where $w_i \in \mathbb{R}^+$ is a weight which models the importance of requirement R_i.

Let us consider the case where (X, c) fails a non-empty set of requirements; this set, given the definitions, is equal to $\mathtt{id}[\mathcal{R}]$. Now, let $\mathcal{K} = \{k_1, \ldots, k_q\}$ be an arbitrary set of spatial transformations independent of $\mathcal{M}$; we say that a spatial transformation $k \in \mathcal{K}$ *repairs* (X, c) if $k[\mathcal{R}] \subset \mathtt{id}[\mathcal{R}]$; we say that k *fully repairs* (X, c) if $k[\mathcal{R}]$ is empty. The set spatial transformations $\mathcal{K}$ corresponds to a repair toolkit, which we generate systematically just like we did with the set of attacks $\mathcal{M}$. We study the process of system repair in more detail in Chapter **??**.

We now consider an analysis that combines both $\mathcal{M}$ and $\mathcal{K}$ as follows. If an attack $m \in \mathcal{M}$ breaks some requirements, is there a *counter attack* k that fully repairs $(X, c \circ m)$? If so, we can improve the robustness of the system (X, c) by transforming it into $(X, c \circ m \circ k)$ whenever we detect that the attack m is affecting (X, c); we refer to this dependent notion of robustness as *latent robustness*, and we explore this concept in cyber-physical systems during Chapter **??**.

Example 2.3 (Counter Attack)**.** In the context of Example 2.1, consider the behavioural property $Q\colon 2^{2^*} \to 2$ which accepts a language $\phi \in 2^{2^*}$ if and only if ϕ accepts a sequence $w \in 2^*$, then w ends in 1 or w is empty.

In the original F-coalgebra, all states in 2×2 satisfy the property Q since both $(0, 0)$ and $(1, 0)$ recognise the language $(0|1)^*11$, $(0, 1)$ recognises the language $(0|1)^*1$, and $(1, 1)$ recognises $\varepsilon + 0(0|1)^*11 + 1(0|1)^*1$. However, in the latent coalgebra revealed by the attack m, which maps (a, b) to $(\neg a, \neg b)$, the state $(0, 0)$ no longer satisfies Q, since it now recognises the language $\varepsilon|(0|1)^*10$.

Now, if we use m again in the latent coalgebra $(2\times 2, c\circ m)$ to reveal the latent coalgebra $(2\times 2, c\circ m\circ m)$, we repair the system with respect to the property Q, since $m\circ m = \mathtt{id}$. Thus, m can be used as its own counter attack, but only when the latent coalgebra $(2\times 2, c\circ m)$ has been revealed (otherwise, the application of m reveals $(2\times 2, c\circ m)$, and it breaks Q).

2.5.2 Classification of Attackers

We have not discussed how we systematically generate attacks and counter attacks for an F-coalgebra (X, c). Both attacks and counter attacks are spatial transformations, so they are elements of the set X^X. Our goal is the following: given an *attacker model* (whatever that is), we want to automatically obtain a set of attacks $\mathcal{M}$, which we can use to perform LBA on the F-coalgebra (X, c). We can use the results of this analysis to quantify robustness as shown in Chapter **??**. In this section, we are interested in also varying the attacker model, and in comparing them.

Consider an F-coalgebra $(\vec{X}, c)$ such that $\vec{X}$ is a product type of finite types; e.g., $\vec{X} = 2\times 2$. We use the coordinates of $\vec{X}$, e.g., $\mathtt{fst}$ and $\mathtt{snd}$ to define attacker models.

Definition 2.5.3 (Invariant Attacker Model)**.** Let $(\vec{X}, c)$ be an F-coalgebra such that $\vec{X}$ is a product type whose coordinates are $\Pi = \{\pi_1, \ldots, \pi_n\}$ where $\pi_i \colon \vec{X} \to A_i$; an *invariant attacker model* α is a mapping which takes each coordinate $\pi_i \in \Pi$ to a set of transformations $\mathcal{T}_i \subseteq A_i^{A_i}$ such that the following conditions are satisfied:

- the attacker can always do nothing, i.e., $\mathtt{id}_{A_i} \in \mathcal{T}_i$;
- if the attacker can transform the coordinate π_i using $f \in \mathcal{T}_i$, and they can transform π_i using $g \in \mathcal{T}_i$, then they should be able to transform π_i using $g \circ f$; i.e., if $f, g \in \mathcal{T}_i$, then $g \circ f \in \mathcal{T}_i$.

For a coordinate $\pi \in \Pi$, a transformation $f \in \alpha(\pi)$ acts on a state $\vec{x} \in \vec{X}$ by substituting the value of coordinate π_i at state $\vec{x}$ with $f(x[\pi_i])$; i.e., we can extend f to a function of type $f^\flat \colon X \to X$ defined by

$$f^\flat(\vec{x})[\pi_j] \triangleq \begin{cases} f(\vec{x}[\pi_j]), & \text{if } i = j; \\ \vec{x}[\pi_j], & \text{otherwise.} \end{cases}$$

The natural extensions from $f \in \alpha(\pi)$ to $f^\flat \in \vec{X}^{\vec{X}}$ lifts the attacker model α into a set of generator functions for the set $\langle\alpha\rangle \subseteq \vec{X}^{\vec{X}}$, where $\langle\alpha\rangle$ is smallest set satisfying the following conditions:

- for all $\pi \in \Pi$ and $f \in \alpha(\pi)$, $f^\flat \in \langle\alpha\rangle$;
- for all $f, g \in \langle\alpha\rangle$, $g \circ f \in \langle\alpha\rangle$.

Example 2.4 (Some Invariant Attacker Models for Example 2.1)**.** Since $\vec{X} = 2\times 2$, we have two coordinates: $\mathtt{fst} \colon \vec{X} \to 2$ and $\mathtt{snd} \colon \vec{X} \to 2$ mapping $(x, y) \xmapsto{\mathtt{fst}} x$ and $(x, y) \xmapsto{\mathtt{snd}} y$. The trivial attacker model is an attacker that does nothing, and it corresponds to the attacker model $\mathtt{id}$ which maps $\mathtt{fst}$ to $\{\mathtt{id}_2\}$ and $\mathtt{snd}$ to $\{\mathtt{id}_2\}$. The lifting of these functions yield the set $\langle\mathtt{id}\rangle = \{\mathtt{id}_2^\flat\}$ with $\mathtt{id}_2^\flat = \mathtt{id}_X$.

We consider three other attackers now: one which has full control over `fst` but no control over `snd`, one which has full control over `snd` but not over `fst`, and one attacker that has full control over both `fst` and `snd`. The attacker with no control over `fst` or `snd` is the attacker which generates $\langle \mathtt{id} \rangle$.

The attacker with full control over `fst`, denoted α_1, maps `snd` to $\{\mathtt{id}_2\}$ and `fst` to 2^2. The set $\langle \alpha_1 \rangle$ is the set of functions in $\vec{X}^{\vec{X}}$ which only affect the `fst` coordinate. Similarly, the attacker with full control over `snd`, denoted α_2, maps `fst` to $\{\mathtt{id}_2\}$ and `snd` to 2^2. Dually, the set $\langle \alpha_2 \rangle$ is the set of functions in $\vec{X}^{\vec{X}}$ which only affect the `snd` coordinate. We study these type of attackers which invariantly affect only one coordinate in Chapter **??**.

There are attackers with partial control over `fst` and `snd`; e.g., an attacker β which maps `fst` to $\{\Delta_1, \mathtt{id}\}$ (Δ_1 is the constant function that maps $b \in 2$ to 1), which generates $\langle \beta \rangle = \{\Delta_1^\flat, \mathtt{id}_2^\flat\}$, where

$$\Delta_1^\flat(\vec{x})[\mathtt{fst}] = 1, \quad \text{and} \quad \Delta_1^\flat(\vec{x})[\mathtt{snd}] = \vec{x}[\mathtt{snd}].$$

Note that $\langle \beta \rangle \subseteq \langle \alpha_1 \rangle$. Henceforth we only consider attackers which have full control over a particular set of coordinates since we did not find any practical scenarios where restricting the control of attackers over a particular coordinate offered a significant gain over the attacker with full control over such a coordinate.

The attacker with full control over `fst` and `snd`, denoted α^*, can affect both `fst` and `snd` at the same time. However, this does not mean that $\langle \alpha^* \rangle$ is equal to $\vec{X}^{\vec{X}}$, since there are still functions that cannot be generated by the combination of individual effects on independent coordinates. For example, with $\vec{X} = 2 \times 2$, consider the function $m \colon \vec{X} \to \vec{X}$ where $m(x, y) \triangleq (y, x)$; this function cannot be written in terms of a product function $f \times g \colon 2 \times 2 \to 2 \times 2$, with $f \colon 2 \to 2$ and $g \colon 2 \to 2$. In Chapter **??**, we add a more flexibility to attacker models so that they are no longer invariant, and we can generate arbitrary functions in $\vec{X}^{\vec{X}}$.

Given an F-coalgebra $(\vec{X}, c)$ where $\vec{X}$ is a product type of finite types whose coordinates are $\Pi = \{\pi_1, \ldots, \pi_p\}$, and a list of requirements $\mathcal{R} = \{R_1, \ldots, R_n\}$, we can systematically generate invariant attacker models by choosing a subset of coordinates in Π. Under this formulation, the set of invariant attacker models is characterised by $\mathscr{P}(\Pi)$. Given that attacker models are now modelled by sets, we can compare them via set inclusion. This comparison sorts attackers by *capabilities*; i.e., it classifies attackers with respect to the ways they can interact with the system. Now, using LBA, we can obtain the set of requirements $\langle \alpha \rangle[\mathcal{R}]$ that the attacker model α can break, and we can compare these sets of requirements by set inclusion to compare attackers by their *power*; i.e., for two attacker models α and β, if $\langle \alpha \rangle[\mathcal{R}] \subset \langle \beta \rangle[\mathcal{R}]$, then β breaks strictly more requirements than α, making β a more dangerous adversary than α.

2.5.3 Side-Channel Repair

The last application of LBA that we consider is of program repair. We briefly mentioned the idea in Section 2.5.1: given an F-coalgebra (X, c) and a set of requirements $\mathcal{R} =$

$\{R_1, \ldots, R_n\}$, if $\texttt{id}[\mathcal{R}]$ is non-empty, then we can look for spatial transformations to reveal a latent behaviour $(X, c \circ m)$ such that $m[\mathcal{R}]$ is empty. We consider an interesting case where the state x is a program, and R consists of a single requirement: constant memory access patterns. We propose an idempotent spatial transformation $\mathcal{O}$ which maps x to a state which satisfies constant memory access patterns. In Chapter **??**, we show how we can use $\mathcal{O}$ to repair a family of timing side-channel leaks for LLVM programs.

Appendix A

Appendix Title Here

Write your Appendix content here.

Bibliography

Shacham, Hovav (2007). "The Geometry of Innocent Flesh on the Bone: Return-into-Libc without Function Calls (on the X86)". In: *Proceedings of the 14th ACM Conference on Computer and Communications Security*. CCS '07. Alexandria, Virginia, USA: Association for Computing Machinery, 552–561. ISBN: 9781595937032. DOI: 10.1145/1315245.1315313. URL: https://doi.org/10.1145/1315245.1315313.