Testing Autonomous and Automated Driving Functions

Challenges and Potential Solutions

Prof. Dr. Franz Wotawa,
TU Graz, Institute for Software Technology, Christian Doppler Laboratory for Quality Assurance Methodologies for Autonomous Cyber-Physical Systems (QAMCAS)
wotawa@ist.tugraz.at

1

Thanks go to

- Jianbo Tao, Florian Klück, Lorenz Klampfl, Mihai Nica, Herman Felbinger, Yihao Li, Martin Zimmermann, and many others for their contributions over the last years in the context of AD/ADAS testing
- Funding agencies and their corresponding projects

2

Content

- **Motivation** – Why is it important to assure safety in the context of autonomous driving (AD) and ADAS?
- **Ontology-based testing** – Using ontologies and combinatorial testing for identifying critical scenarios
- **Search-based testing** for AD/ADAS – Using genetic algorithms for extracting test cases
- **Comparison of methods**
- **Conclusion**

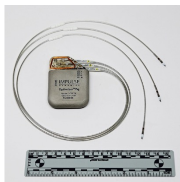
3

Safety-critical systems

A safety-critical system is a system whose failure or malfunction may result in one (or more) of the following outcomes:

- death or serious injury to people
- loss or severe damage to equipment/property
- environmental harm

J. C. Knight, "Safety critical systems: challenges and directions," Proceedings of the 24th International Conference on Software Engineering. ICSE 2002, Orlando, FL, USA, 2002, pp. 547-550.

4

Important Aspects for Testing AD/ADAS

- Demonstrate reliability and safety:
  - Safer than human driving to gain public acceptance
  - Statistics (U.S. Department of Transportation, National Highway Traffic Safety Administration):
    - 36,096 deaths on road per year (USA,2019)
    - 1.11 fatalities per 100 Mio vehicle miles (USA, 2019)
  - >90% caused by humans' driving behavior
- Ensure safety in any situation:
  - Master standard situations and corner cases
  - No or just limited human back-up
  - Fail-operational (HW/SW redundancy)
- Manage system complexity:
  - No "pure mechanical system"
  - Highly intervened SW and HW components
  - Sensor false-positive, sensor noise, etc.

5

Challenges for Testing AD/ADAS

- Complete software testing of complex systems?
  - Close to infinite parameter space
  - *"Testing is the process of executing a program with the intention of finding errors"*
- How to assure "good-enough" testing?
  - 275 million miles required to demonstrate safe driving
  - Physically collecting millage is infeasible
  - Unlikely to cover a sufficient amount of critical situations

→ Virtual testing methodology required to:
  - Consider all environmental parameter interactions that influence the AD/ADAS system
  - Identify and reproduce critical scenarios as basis for ADAS Verification and Validation

6

„The family of a San Mateo man who was killed when his Tesla crashed into a gore point at the Highway 101 and Highway 85 connector in Mountain View on March 23,2018. Members of the man's family say he complained to Tesla about issues with the vehicle several times prior to the accident that took his life."

7

„On March 18th 2018, a 49-year-old woman was struck by a self-driving Uber vehicle in Tempe, Arizona. She was transported to the hospital, where she died. In the aftermath, Uber's self-driving program is hanging on by a thread, while the rest of the industry debates the speed in which these vehicles were being rushed to market. It is widely seen as the first person to be killed by an autonomous vehicle."

8


Exterior View

9


The case of airplane automation Boeing 737 Max
„Lion Air 610 crashed because a faulty sensor erroneously reported that the airplane was stalling. The false report of a stall triggered an automated system that tried to point the aircraft's nose down so that it could gain enough speed to fly safely. The pilots fought the automated system, trying to pull the nose back up."
(MIT TR, March 2019)
C-FNAX
Lion Air Flight 610 crashed October 2018 (189 passengers killed)
Ethiopian Airlines Flight 302 crashed March 2019 (157 passengers killed)
© Acefitt, 2018, CC-BY-SA-4.0

10

Tesla accident in Taiwan, 2020.

11

Crash influencing factors

1. There was a bridge before the truck
2. The highway lane is curved
3. The truck has a white color
4. The driver of the Tesla car was not reacting

12

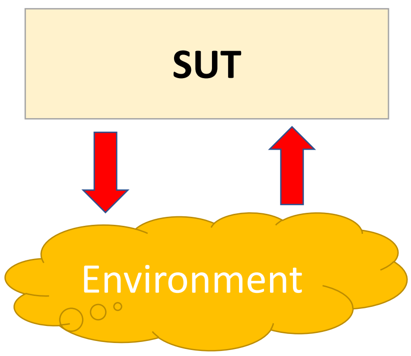
Summary motivation

- Often there is more than one influencing factor for a crash
  - Multiple decisions, interactions, etc. must occur at the same time
- There is a need for testing such scenarios
  - FOCUS on CRITICAL SCENARIOS considering interactions
- Need testing approaches that provide scenarios interacting with the system under test, i.e., the AD car / ADAS function

13

Summary motivation (cont.)

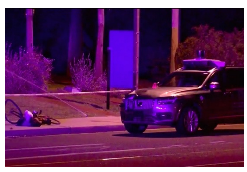
- Need to come up with critical scenarios for safety-critical systems



Scenario: Sequence of interactions

Interaction: Action from SUT + reaction from environment

WANT TO REVEAL A FAULT!!!!



14

Scenario-based ADAS/AD V&V

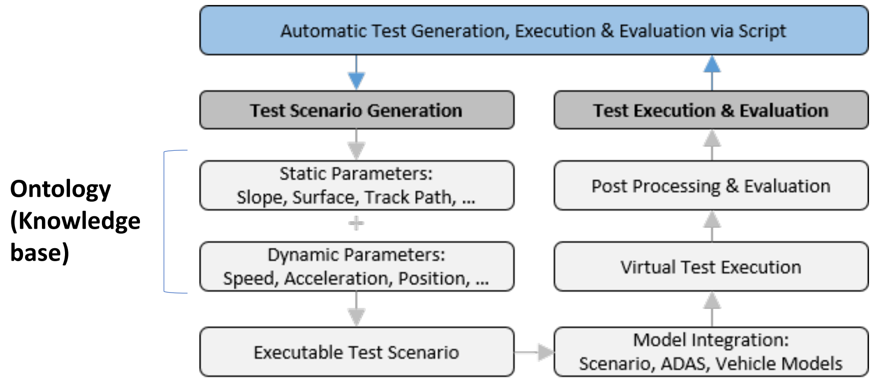
Scenario-based approaches are considered as proper methods

- In safety of the intended Functionality (**SOTIF**) ISO PAS 21448: Use case/Scenario-based Hazard and Risk evaluation–V&V accordingly: **simulation**, real world confirmation;

Figure 11: Example of optical illusion drawing that could fool a vision system.
Source: SOTIF **ISO PAS 21448**

15

Scenario-based testing

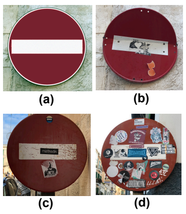


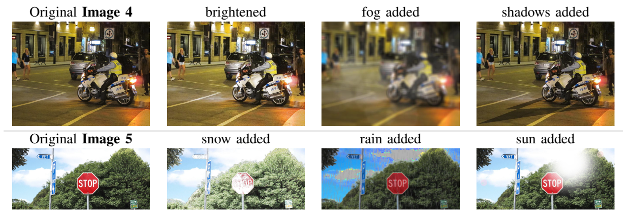
Make us of combinatorial testing for test case generation

16

Have to consider a lot of scenarios!

- **Static part**: streets, traffic signs, houses, marks on street, weather conditions,…





- **Dynamic part**: pedestrians, other cars, indention of ego vehicle,… → **behavior** over time

USE ONTOLOGIES FOR REPRESENTING KNOWLEDGE

17

Critical scenarios identification for ADAS/AD systems

- What is safety critical scenario?
- How to find and identify critical scenarios?

Finding Critical Scenarios for Automated Driving Systems: A Systematic Mapping Study

Xinhai Zhang, Jianbo Tao, Kaige Tan, *Member, IEEE,* Martin Törngren, *Senior Member, IEEE,* José Manuel Gaspar Sánchez, *Member, IEEE,* Muhammad Rusyadi Ramli, *Member, IEEE,* Xin Tao, *Member, IEEE,* Magnus Gyllenhammar, Franz Wotawa, *Member, IEEE,* Naveen Mohan, *Member, IEEE,* Mihai Nica, *Member, IEEE,* and Hermann Felbinger, *Member, IEEE,*

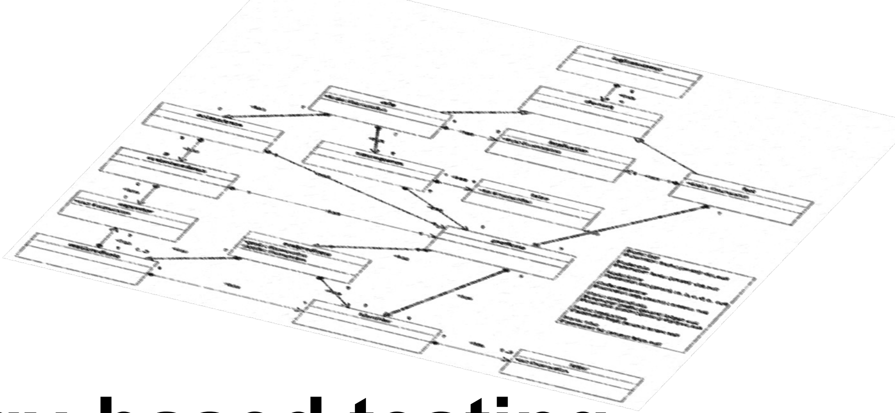
Abstract—Scenario-based approaches have been receiving a huge amount of attention in research and engineering of automated driving systems. Due to the complexity and uncertainty of the driving environment, and the complexity of the driving task itself, the number of possible driving scenarios that an Automated Driving System or Advanced Driving-Assistance System may encounter is virtually infinite. Therefore it is essential to be able to reason about the identification of scenarios and in particular critical ones that may impose unacceptable risk if not considered. Critical scenarios are particularly important to support design, verification and validation efforts, and as a basis for a safety case. In this paper, we present the results of a systematic mapping study in the context of autonomous driving. The main contributions are: (i) introducing a comprehensive taxonomy for critical scenario identification methods; (ii) giving an overview of the state-of-the-art research based on the taxonomy encompassing 86 papers between 2017 and 2020; and (iii) identifying open issues and directions for further research. The provided taxonomy comprises three main perspectives encompassing the problem definition (the why), the solution (the methods to derive scenarios), and the assessment of the established scenarios. In addition, we discuss open research issues considering the perspectives of coverage, practicability, and scenario space explosion.

Index Terms—Critical Scenario, Automated Driving, Systematic Mapping Study.

- *X. Zhang is with Sigma Technology Consulting AB and the autonomous group of Scania CV AB in Sweden. He was with the Mechatronics division of KTH for most of the time when writing this paper.*
  E-mail: xinhai@kth.se
- *J. Tao, M. Nica and H. Felbinger are with AVL.*
- *K. Tan, M. Törngren, J. Gaspar, M. R. Ramli, M. Gyllenhammar and N. Mohan are with the Mechatronics division at KTH, Stockholm, Sweden*
- *X. Tao is with the Integrated Transport Research Lab (ITRL) of KTH.*
- *M. Gyllenhammar is with Zenseact AB, Gothenburg, Sweden*
- *F. Wotawa is with the CD Laboratory for Quality Assurance Methodology for Autonomous Safety Critical Systems (QAMCAS), Inst. for Software Technology, TU Graz, Graz, Austria*

***Paper**: X. Zhang, J. Tao et al., "Finding Critical Scenarios for Automated Driving Systems: A Systematic Mapping Study," in IEEE Transactions on Software Engineering, doi:* 10.1109/TSE.2022.3170122.

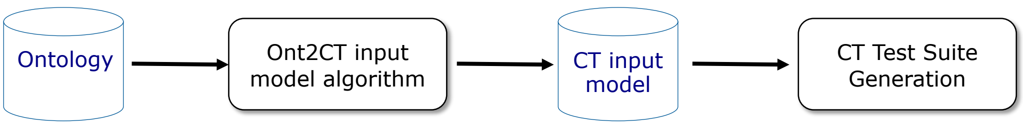
18



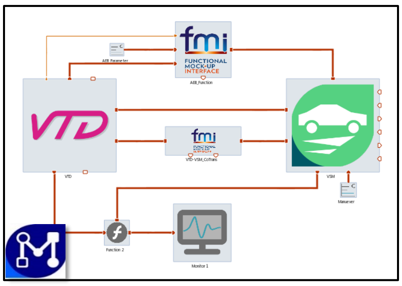
Ontology-based testing

19

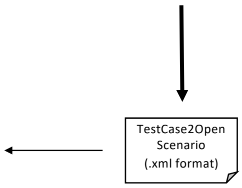
From ontologies to simulation-based testing



- Tool-chain of ontology-based test suite generation approach: from ontology to simulation
- Using **combinatorial testing** approach: apply combinatorial testing method into tool-chain for test scenario creation

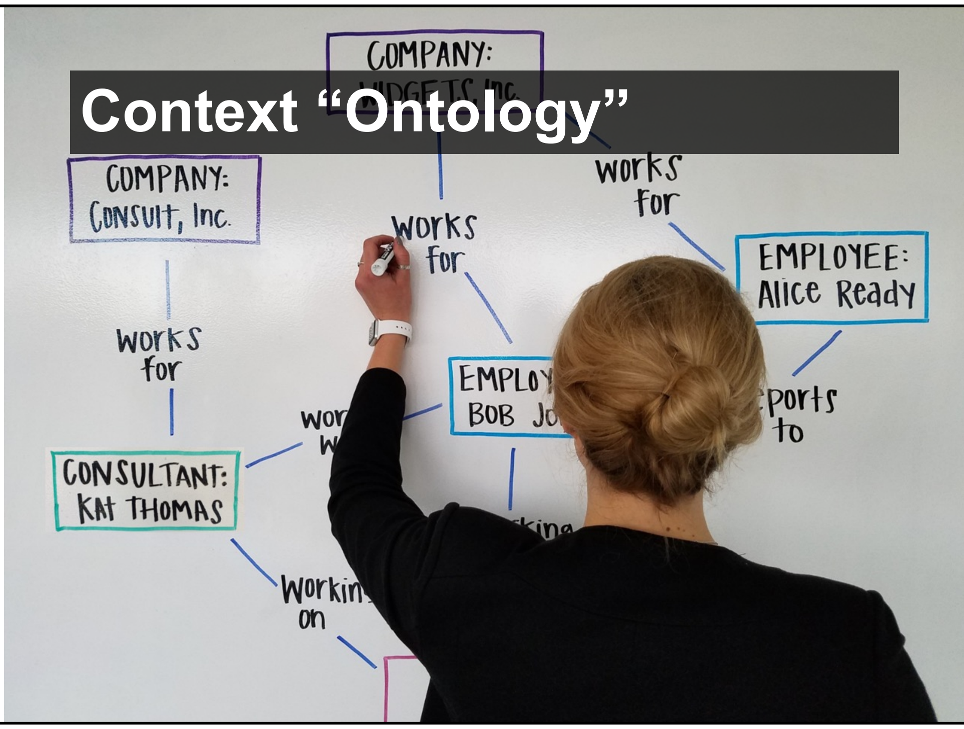


CT Test Execution/Simulation



20

Context “Ontology”



21

Regarding Ontology…

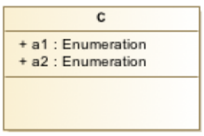
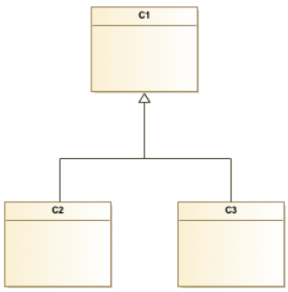
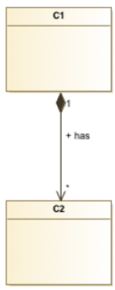
- *An ontology is a formal, explicit specification of a shared conceptualization that is characterized by high semantic expressiveness required for increased complexity*

Feilmayr, C., Wöß, W.: An analysis of ontologies and their success factors for application to business. Data & Knowledge Engineering pp. 1{23 (2016). https://doi.org/10.1016/j.datak.2015.11.003

22

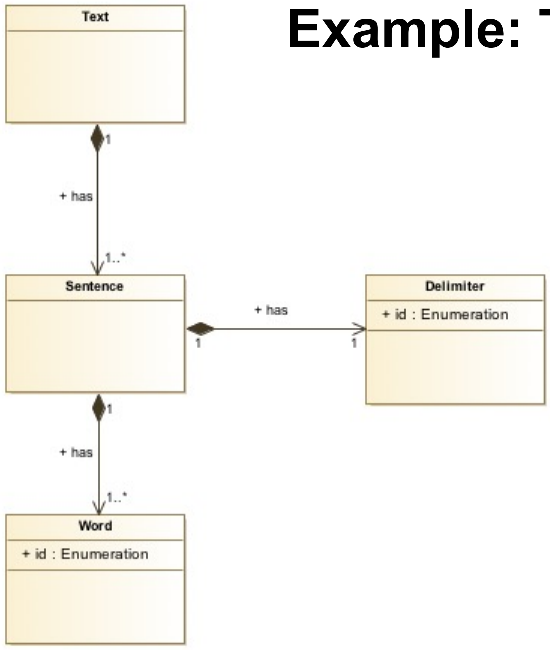
- To describe concepts in a formal way
  - A concept describes an entity either from the real world (e.g., a car) or from nonmaterial descriptions (e.g., a sentence or a physical force)
- To describe the knowledge (e.g., relationships) of these concepts
  - Attributes
  - Inheritance
  - Composition

23

		
Attributes: Concept C with two attributes $a1$, $a2$ of basic type `Enumeration`.	*Inheritance:* Concepts $C2$ and $C3$ are sub-concepts of $C1$.	*Composition:* Concept $C1$ comprises instances of concept $C2$. There might be 0 to n concepts $C2$ assigned to $C1$ depending on the value of $*$.

24

Example: Text ontology



25

Definition 1: Ontology

- An **ontology** is a tuple (C, A, D, ω, R, τ, ψ) where:

C is a finite set of concepts

A is a finite set of attributes

D is a finte set of domain elements

$\omega : C \mapsto 2^{A \times 2^D}$ *is a function mapping concepts to a set of tuples specifying the attribute and its domain elements*

R *is a finite set of tuples from C×C stating that two concepts are related*

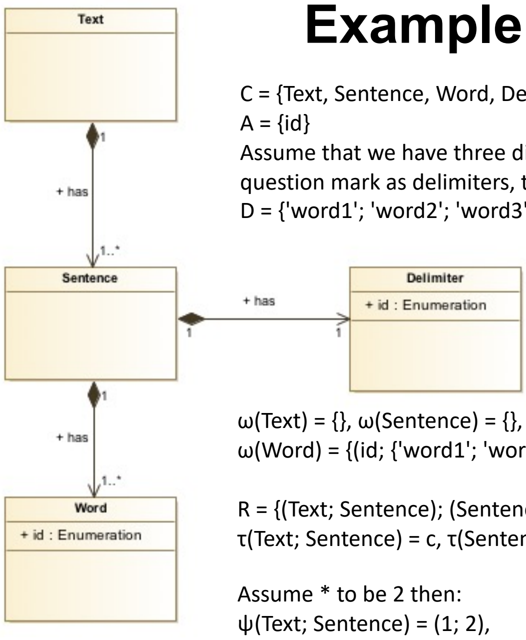
$\tau : R \times R \mapsto \{c, i\}$ *assigns a type to each relation using* ***i*** *for* ***inheritance*** *and* ***c*** *for* ***composition***

$\psi : R \times R \mapsto IN_0 \times IN_0$ *is a function mapping relationships solely for type c to its minimum and maximum arity.*

The ***arity*** *is for specifying how many concepts a particular concepts may comprise and ranges from 0 to any arbitrary natural number*

26

Example: Text ontology



C = {Text, Sentence, Word, Delimiter}
A = {id}
Assume that we have three different words and only a period and a question mark as delimiters, then:
D = {'word1'; 'word2'; 'word3'; '.'; '?'}

ω(Text) = {}, ω(Sentence) = {}, ω(Delimiter) = f{(id; {'?'; '.'})},
ω(Word) = {(id; {'word1'; 'word2'; 'word3'})}

R = {(Text; Sentence); (Sentence; Delimiter); (Sentence; Word)}.
τ(Text; Sentence) = c, τ(Sentence; Delimiter) = c,τ(Sentence; Word) = c.

Assume * to be 2 then:
ψ(Text; Sentence) = (1; 2),
ψ(Sentence; Delimiter) = (1; 1),
ψ(Sentence; Word) = (1; 2).

27

Definition 2: Root Concept & Leaf Concept

- Given an ontology is a tuple (*C*, *A*, *D*, *ω*, *R*, *τ*, *ψ*), a concept c is a **root concept** if and only if there exists no relation $(\mathrm{c'}, \mathrm{c}, \mathrm{x}) \in \mathrm{R}$ for $\mathrm{c'} \in C, x \in \{i, c\}$
  - *e.g., Text*

- A concept c is a **leaf concept** if and only if there is no relation $(\mathrm{c}, \mathrm{c'}, \mathrm{x}) \in \mathrm{R}$ for $\mathrm{c'} \in C, x \in \{i, c\}$
  - *e.g., Sentence*

28

Definition 3 (Cyclic ontology). *An ontology* $(C, A, D, \omega, R, \psi)$ *is called a* cyclic ontology *if and only if (i) there is a relation* $(c, c, x) \in R$ *for a concept* $c \in C$ *and* $x \in \{\mathtt{i}, \mathtt{c}\}$*, or (ii) there are relations* $(c_0, c_1, x_0), \ldots, (c_i, c_0, x_i)$ *in* R *for* $i > 0$*, concepts* $c_0, \ldots, c_i inC$ *and* $x_0, \ldots, x_i \in \{\mathtt{i}, \mathtt{c}\}$*. In this case the sequence of relations is called a cycle. If an ontology is not cyclic, it is called acyclic ontology.*

Definition 4 (Well-formed ontology). *An ontology* $(C, A, D, \omega, R, \psi)$ *is a* well-formed ontology *if and only if (i) it comprises exactly one root concept, (ii) it is acyclic, and (iii) where all its leaf concepts have attributes.*

29

Parameters

P1 : A , B , C
P2 : 1 , 2
P3 : X , Y

All Combinations

		P1	P2	P3
TC1	:	A	1	X
TC2	:	A	1	Y
TC3	:	A	2	X
TC4	:	A	2	Y
TC5	:	B	1	X
TC6	:	B	1	Y
TC7	:	B	2	X
TC8	:	B	2	Y
TC9	:	C	1	X
TC10	:	C	1	Y
TC11	:	C	2	X
TC12	:	C	2	Y

2-Pair (Pairwise)

		P1	P2	P3
TC1	:	A	1	X
TC4	:	A	2	Y
TC6	:	B	1	Y
TC7	:	B	2	X
TC9	:	C	1	X
TC12	:	C	2	Y

30

- *A method that aims to improve the effectiveness of software testing while lowering its cost at the same time. The essence of CT is that not all parameters contribute to failures but by interactions between relatively few parameters.*

Kuhn, D.R., Kacker, R.N., Lei, Y.: Combinatorial testing. In: Laplante, P.A. (ed.) Encyclopedia of Software Engineering. Taylor & Francis (2012)

- In combinatorial testing we search for all tests that cover all combinations for any subset of size ***k*** of the variables, where ***k*** is called the ***strength*** of the generated test suite.

31

Definition 5: Combinatorial Testing Input Model

- A **combinatorial testing input model** is a tuple (V, DOM, CONS) where:
  - **V** is a set of variables
  - **DOM** is a function mapping variables from V to a set of values
  - **CONS** is a set of constraints over variables that have to be fulfilled for each test case

32

Example:

- Assume that there is an application which needs to be run on different platforms consisting of five components (or parameters):
  - OS (Windows XP, Apple OS X, Red Hat Enterprise Linux),
  - browser (Internet Explorer, Firefox),
  - protocol (IPv4, IPv6),
  - CPU (Intel, AMD),
  - database (MySQL, Sybase, Oracle).

There is a total of 72 (i.e.,3×2×2×2×3) possible platforms (or combinations).

33

- It only requires 10 tests for conducting a 2-way (i.e., *k*=2) or pairwise testing to cover all possible pairs of platform components

Test	OS	Browser	Protocol	CPU	DBMS
1	XP	IE	IPv4	Intel	MySQL
2	XP	Firefox	IPv6	AMD	Sybase
3	XP	IE	IPv6	Intel	Oracle
4	OS X	Firefox	IPv4	AMD	MySQL
5	OS X	IE	IPv4	Intel	Sybase
6	OS X	Firefox	IPv4	Intel	Oracle
7	RHEL	IE	IPv6	AMD	MySQL
8	RHEL	Firefox	IPv4	Intel	Sybase
9	RHEL	Firefox	IPv4	AMD	Oracle
10	OS X	Firefox	IPv6	AMD	Oracle

34

From Ontology to CT input model

Ontology
a formal conceptualization of entities, their interfaces and behaviors, and relationships, describe the knowledge behind

Widely used for different application domains. E.g., making, describing traffic situations and navigations in the context of autonomous driving

Conversion

Provides information about certain entities and their relationships to extract test cases

CT input model
takes a set of variables representing inputs and parameters together with their domains as input model

Wildly used test generation technique

Combinatorial Testing

Effective software testing for autonomous driving!

35

Ontology Conversion

- Given an ontology (C, A, D, ω, R, τ, ψ), we describe the conversion for the following three cases:
  - **concepts with attributes**
  - **inheritance relations**
  - **compositional relations**

- A combinatorial testing input model M^{CT} comprising its variables, their domains, and constraints denoted by V^{CT} , DOM^{CT} , and $CONS^{CT}$ respectively, i.e., $\mathbf{M^{CT} = (V^{CT} ;DOM^{CT} ;CONS^{CT})}$.

- A combinatorial testing algorithm is denoted as **CT(M, t)** where M is a combinatorial testing input model and t the combinatorial strength, returning a test suite.

36

Algorithm 1

Algorithm 1 TC_GEN (O,t)

Require: *A well-formed ontology O and a combinatorial strength t.*
Ensure: *A combinatorial test suite for the root concept of the ontology O.*

Let r be the root concept of ontology O.
Call **CT_ONT**(r, O, t) and store the result in $(V^{CT}, DOM^{CT}, CONS^{CT})$.
return $CT((V^{CT}, DOM^{CT}, CONS^{CT}), t)$

40

Algorithm 2

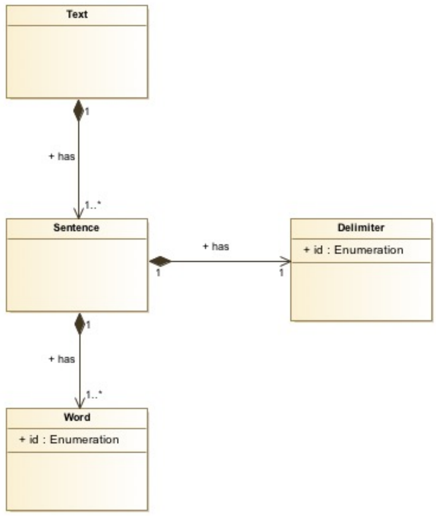
Algorithm 2 CT_ONT (c,O,t)

Require: *A concept c of a well-formed ontology O, and a combinatorial strength k.*
Ensure: *A combinatorial input model for n.*

Let V^{CT} and $CONS^{CT}$ be empty sets.
for all attributes $a \in \omega(c)$ **do**
    Add c_a to V^{CT}.
    Let $DOM^{CT}(c\_a)$ be $dom(c, a)$.
end for
if c is not a leaf concept **then**
    Let tmp be the empty set.
    for all relations $(c, c') \in R$ with type $\tau(c, c') = \mathtt{i}$ **do**
        Add $CT($**CT_ONT**$(c', O, t), t)$ to tmp
    end for
    if tmp is not empty **then**
        Add c to V^{CT}.
        Let $DOM^{CT}(c)$ be tmp.
    end if
    for all relations $(c, c') \in R$ with type $\tau(c, c') = \mathtt{c}$ **do**
        Add variables c'_1 to c'_m to V^{CT}.
        Let d be $CT($**CT_ONT**$(c', O, t), t) \cup \{\epsilon\}$
        for $i = 1$ to m **do**
            Let $DOM^{CT}(c'\_i)$ be d.
        end for
        if $\psi(c, c') = (1, x)$ **then**
            Add $\bigvee_{i \in \{1,\ldots,m\}} c'\_i \neq \epsilon$ to $CONS^{CT}$.
        end if
    end for
end if
return $(V^{CT}, DOM^{CT}, CONS^{CT})$

41

Example: Text Ontology



42

- 2-way test suite for the instances of concept ***Sentence*** using ACTS3.1 and IPOG

$$M^{CT}(Sentence):$$
$$V^{CT}(Sentence) = \{w1, w2, d\}$$
$$DOM^{CT}(w1) = \{'word1', 'word2', 'word3', \in\}$$
$$DOM^{CT}(w2) = \{'word1', 'word2', 'word3', \in\}$$
$$DOM^{CT}(d) = \{'.', '?', \in\}$$
$$CONS^{CT}(Sentence) = \{w1 \neq \in \vee w2 \neq \in, d \neq \in\}$$

	$w1$	$w2$	d
1	'word1'	'word1'	'?'
2	'word1'	'word2'	'.'
3	'word1'	'word3'	'?'
4	'word1'	ϵ	'.'
5	'word2'	'word1'	'.'
6	'word2'	'word2'	'?'
7	'word2'	'word3'	'.'
8	'word2'	ϵ	'?'
9	'word3'	'word1'	'.'
10	'word3'	'word2'	'?'
11	'word3'	'word3'	'.'
12	'word3'	ϵ	'?'
13	ϵ	'word1'	'.'
14	ϵ	'word2'	'?'
15	ϵ	'word3'	'?'

43

- When using ACTS 3.1 and IPOG, we obtain 255 test cases for the concept **Text** from this input model with combinatorial strength 2.

$M^{CT}(Text):$

$V^{CT}(Text) = \{s1, s2\}$

$DOM^{CT}(s1) = \{('word1','word2','?'), ('word1','word2','.'), ..., (\in,'word3','?'), \in\}$

$(having$ 15 items from the test suite of Sentence and additionally $\in)$

$DOM^{CT}(s2) = \{('word1','word2','?'), ('word1','word2','.'), ..., (\in,'word3','?'), \in\}$

$(having$ 15 items from the test suite of Sentence and additionally $\in)$

$CONS^{CT}(Text) = \{s1 \neq \in \vee s2 \neq \in\}$

- Fewer tests can be obtained if the strength is less than the number of sentences

44

Remarks

- There is a better algorithm for generating CT input model from ontologies

Yihao Li, Jianbo Tao, and Franz Wotawa. Ontology-based Test Generation for Automated and Autonomous Driving Functions. Information and Software Technology, Volume 117, January, 2020. https://doi.org/10.1016/j.infsof.2019.106200

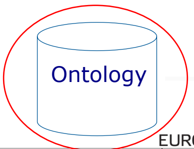
Florian Klück, Yihao Li, Mihai Nica, Jianbo Tao, and Franz Wotawa. Using Ontologies for Test Suites Generation for Automated and Autonomous Driving Functions. In Proc. of the IEEE 29th International Symposium on Software Reliability Engineering (ISSRE-2018) – Industrial Track, Memphis, USA, October 15-18, 2018

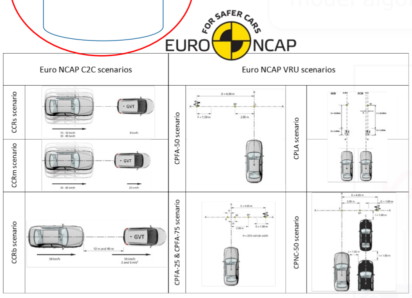
45

Experimental results obtained

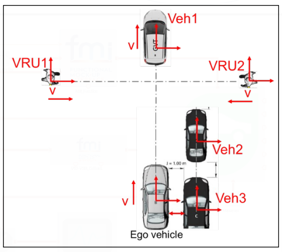
- Testing an AEB function

AEB use case

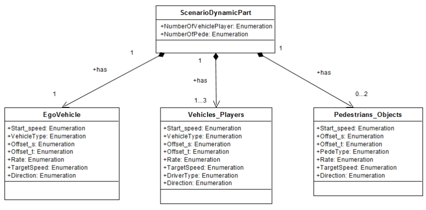




Source: Euro NCAP AEB C2C and VRU protocol



Ontology for AEB based on Euro NCAP protocol using UML

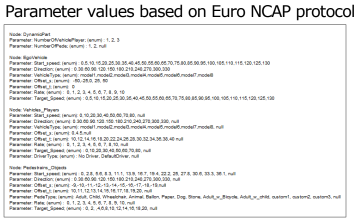


	AEB Pedestrian					
	CPFA-50	CPNA-25	CPNA-75	CPNC-50	CPLA-50	CPLA-25
Type of test	AEB					FCW
VUT speed	20-60 km/h					50-80 km/h
Target speed	8 km/h	5 km/h				
Impact location	50%	25%	75%	50%	50%	25%
Lighting condition	Day	Day & Night		Day	Day & Night	
Vehicle lights (night)		Low beam			High beam	

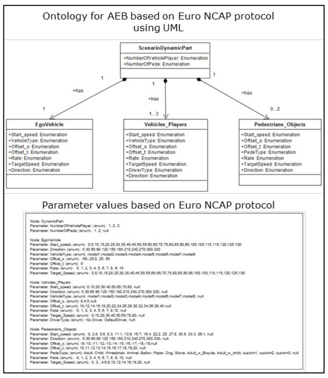
	AEB Bicyclist		
	CBNA-50	CBLA-50	CBLA-25
	AEB		FCW
VUT speed	20-60 km/h	25-60 km/h	50-80 km/h
Target speed	15 km/h		20 km/h
Impact location	50%		25%
Lighting condition	Day		

Source: Euro NCAP AEB C2C and VRU protocol

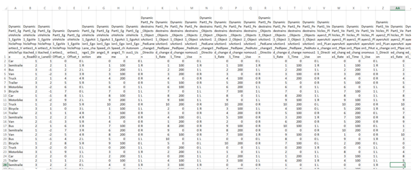
Parameter values based on Euro NCAP protocol



AEB use case



Ont2CT input model algorithm → CT input model → CT Test Suite Generation



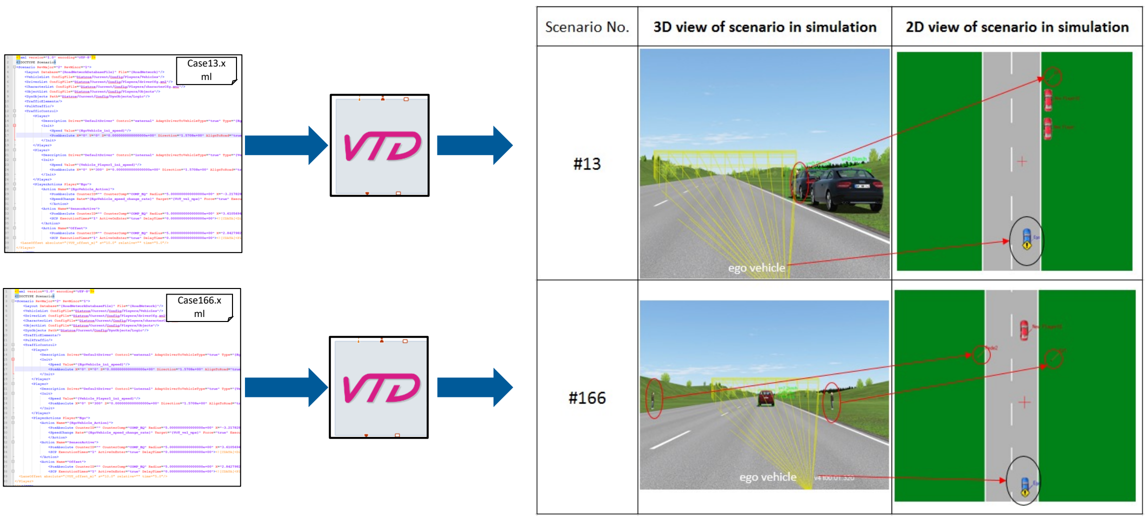
Results with IPOG algorithm:

Strength 2
- Parameters : 39
- Constraints : 42
- Covered Tuples : 71,337
- Number of Tests : 993

Strength 3
- Parameters : 39
- Constraints : 42
- Covered Tuples : 8,163,236
- Number of Tests : 34,634

48

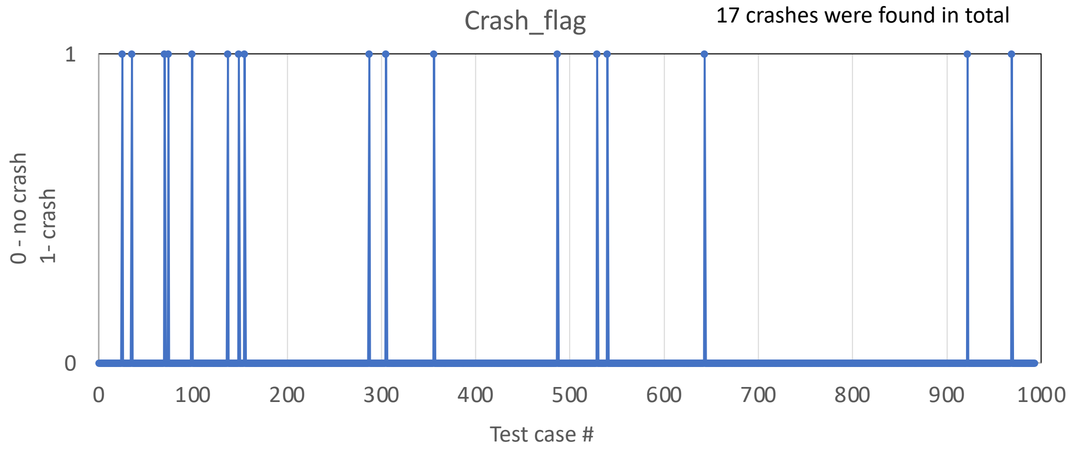
AEB use case



49

Results - # of crash

17 crashes were found in total



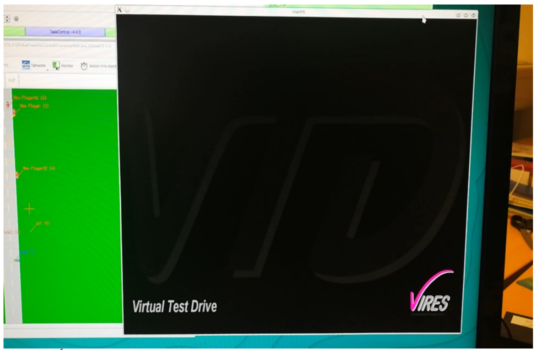
50

Crash case

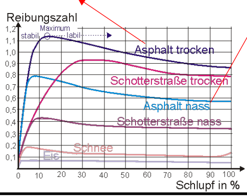
Case with ego vehicle offset to the right!
(#35)

51

Crash case dry vs. wet road



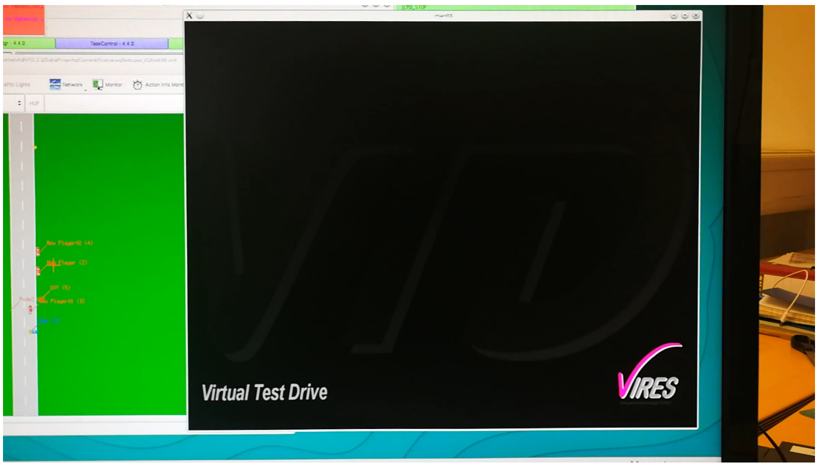
Case No with dry road
No crash (#70)



Case with wet road
crash (#70_wet)

52

Crash case with pedestrians #199



53

Summary & Conclusions

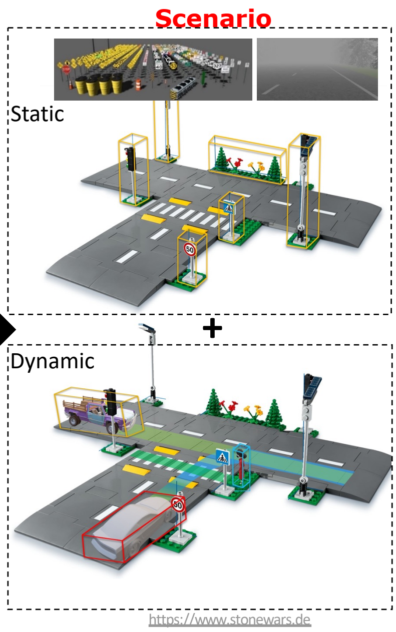
- Ontologies in combination with combinatorial testing can find critical scenarios in the case of autonomous driving & ADAS
- Ontologies describe the environment comprising a static and a dynamic part
- There are standardized ontologies, e.g., OpenX Ontology project of ASAM

54

54

Static and Dynamic Elements & Relationship wrap-up

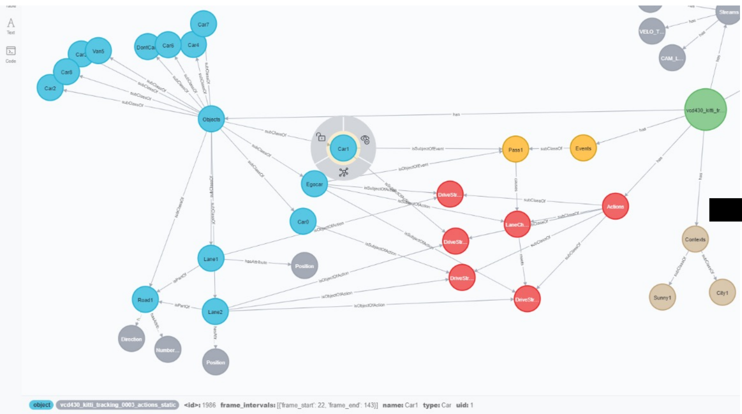
Diverse and complex Situation

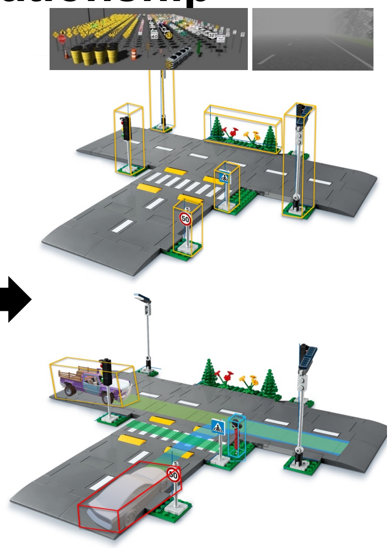


Scenario: The temporal development between several scenes in a sequence of scenes.

55

A general ontology Model for ADAS/AD System Static and Dynamic Elements & Relationship



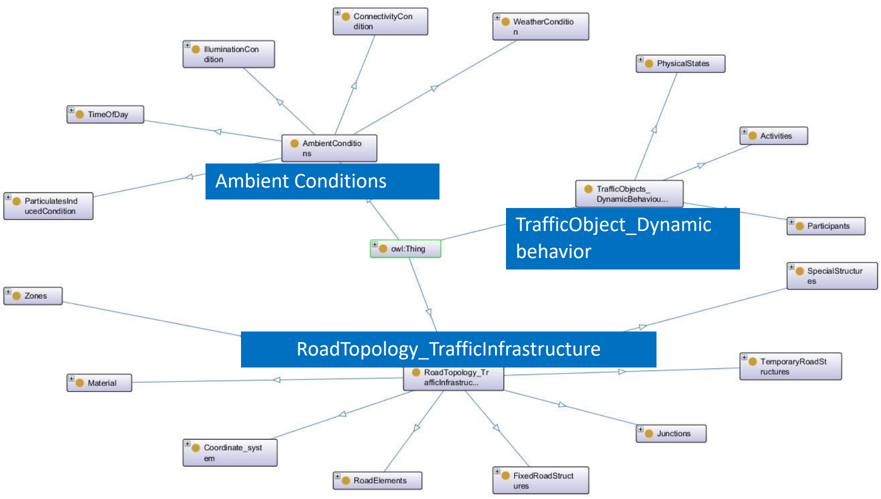


**Ontology: a formal, explicit specification of a shared conceptualization of entities, interfaces, behaviors, and relationships*

56

A general ontology Model for ADAS/AD System

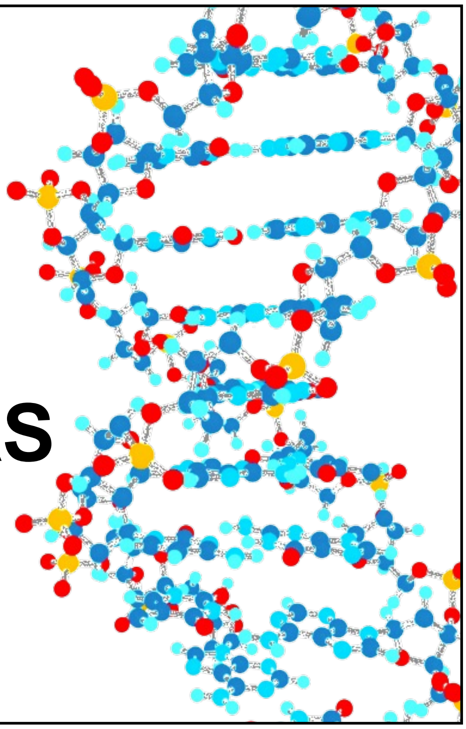
- Ontology model using OWL language (semantic web language based) representing the ODD information (fulfills **ASAM OpenXOntology standard**) for test case generation
- The ontology includes the following entities, their relationships and constraints:
  - ✓ Road infrastructure
  - ✓ Traffic infrastructure
  - ✓ Dynamic traffic participants and their behaviors
  - ✓ Environment information including



OWL: Web Ontology Language

57

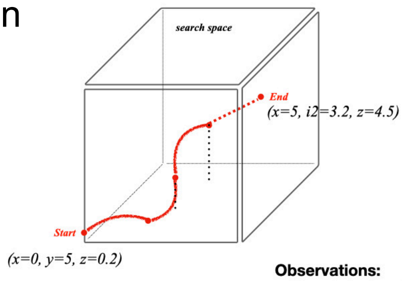
Search-based testing for AD/ADAS



58

Motivation

- Testing, i.e., generating tests revealing bugs, can be seen as search problem



Question: How to go from a start to and end state?

In case of testing: How to find a test suite that (most likely) reveals faults?

Observations:
- Search for input values
- How to justify that a test suite reveals faults?
  → May use coverage or mutation score!

59

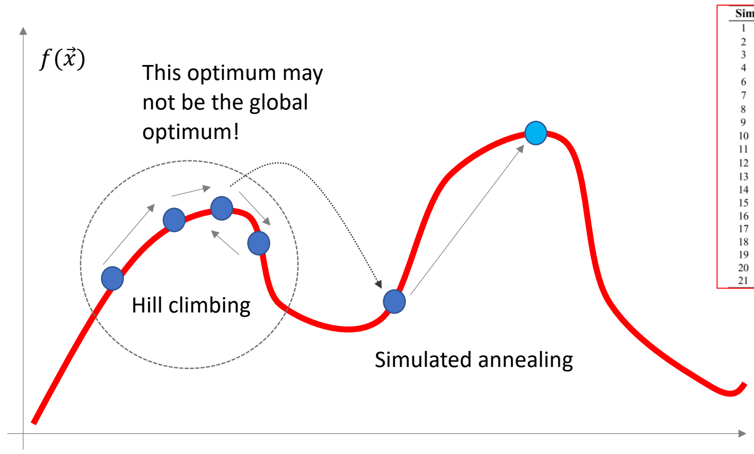
Different search algorithms

- Graph-based search algorithms
  - A* makes use of heuristics function to find a path through a graph
  - Heuristics search as generalization
- Backtracking
- Random search
- Simulated annealing
- Hill climb search
- Genetic programming

Given:
Function $f(\vec{x})$
Wanted:
$\vec{x}$ such that f reaches the optimal value

60

Problems with reaching the optimum



Simulated annealing algorithm

```
Select the best solution vector x_0 to be optimized
Initialize the parameters: temperature T, Boltzmann's constant k, reduction factor c
while termination criterion is not satisfied do
    for number of new solution
        Select a new solution: x_0+Δx
        if f(x_0+Δx) > f(x_0) then
            f_new = f(x_0 + Δx); x_0 = x_0 + Δx
        else
            Δf = f(x_0 + Δx) − f(x_0)
            random r(0, 1)
            if r > exp(−Δf/kT) then
                f_new = f(x_0 + Δx), x_0 = x_0 + Δx
            else
                f_new = f(x_0),
            end if
        end if
        f = f_new
        Decrease the temperature periodically: T = c x T
    end for
end while
```

61

How to use search for test case generation?

- Need:
  1. A vector *x*
  2. Function *f(.)* to be optimized

- Ad 1.: *x* can be the set of test cases of a program or system
- Ad 2.: *f(.)* can be a function that returns the quality of *x*

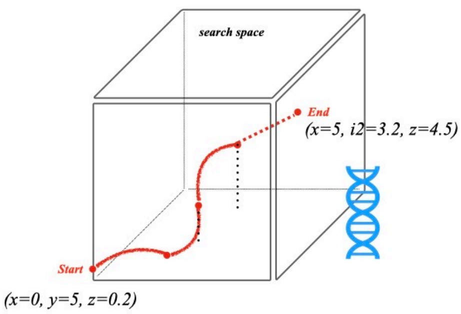
  Make use of coverage and/or mutation score!

62

Genetic programming

- Population where each element has a chromosome
- Apply operators like
  - Selection based on a fitness function
  - Crossover
  - Mutation

to generate new populations

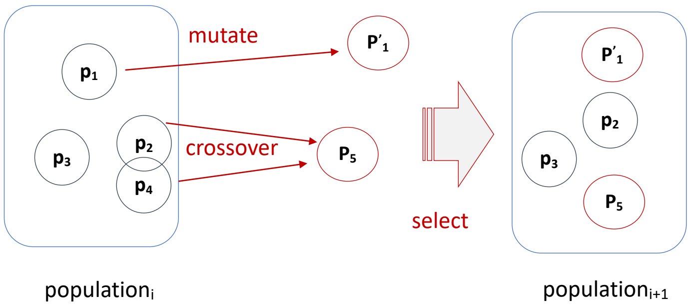


Genetic Search: Using genetic principles to guide search

Questions:
- What are genes?
- What are the operations?

63

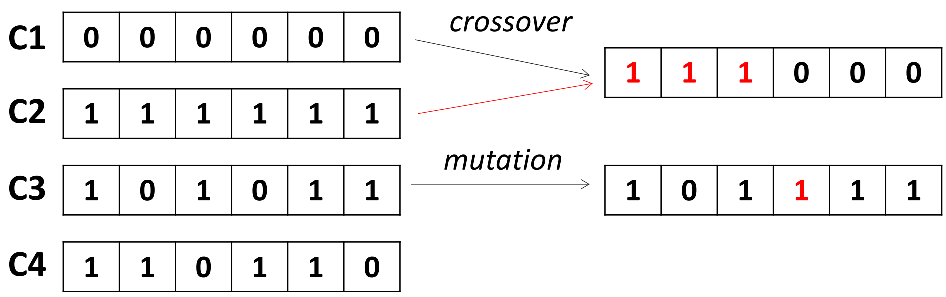
Genetic programming



64

Genetic programming

- Represent problem as chromosomes comprising genes. A set of chromosomes is called a population
- Chromosomes can be stated as strings (or any other collection)



65

Genetic algorithm (GA)

- Crossover:
  - Selection of 2 arbitrary chromosomes
  - Take genes from both to generate a new chromosome
- Mutation:
  - Select 1 arbitrary chromosome
  - Change 1 or more genes for generating a new chromosome
- Selection of chromosomes:
  - Make use of a fitness function

66

Fitness function

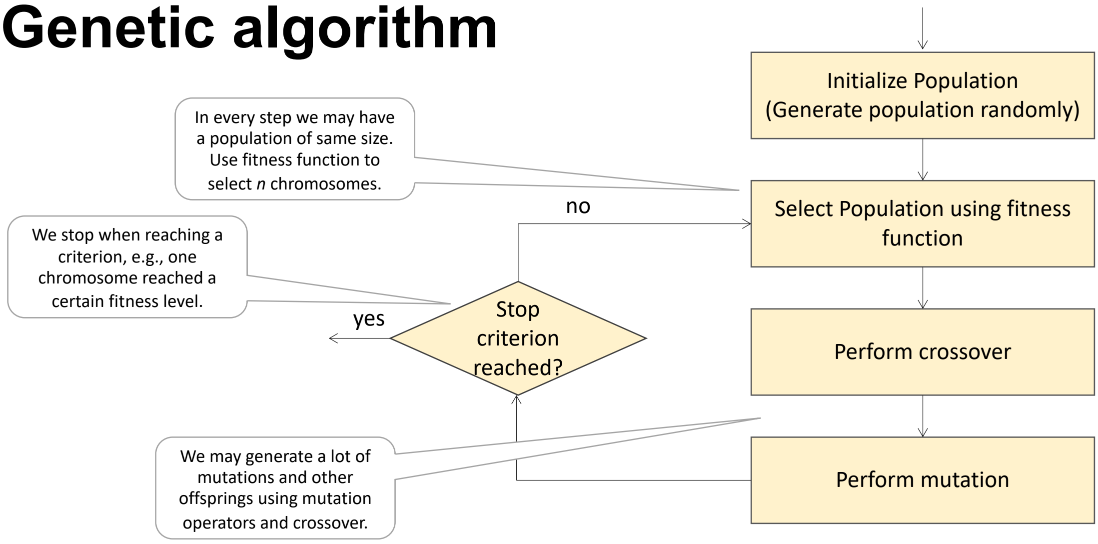
- Maps chromosomes to a particular fitness value

0	**0**	**0**	**0**	**0**	**0**

$\xrightarrow{f(.)}$ *0.923*

67

Genetic algorithm



68

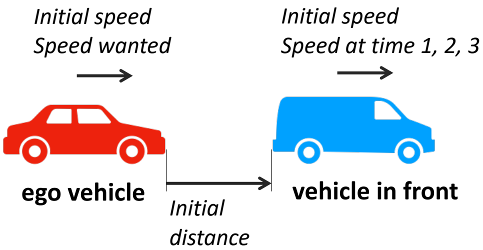
Using GA in the context of testing safety critical systems

- Generating scenarios for testing, e.g., autonomous driving functions
  - Focus on generating interactions between the environment and the system under test
- Generating test data
  - Methods like CT or MBT generate abstract tests, which need to be concretized.
  - Use GA for finding concrete values that more likely lead to revealing faults

69

Generating scenarios

- Have a look at active cruise control (ACC)



Initial speed
Speed at time 1, 2, 3

80	100	25	80	120	70	40

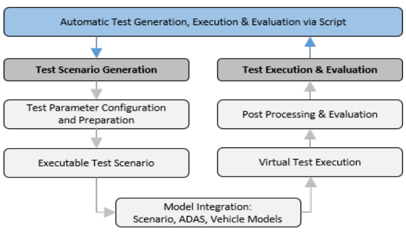
Initial speed Speed wanted Initial distance

- May use mutation function for integers
- Alternatively, we may come up with a collection of values for each gene, we may select. Mutation would be a change of a value for one gene.

70

Another example

- *Using GA for testing an automated emergency braking system (AEB)*
- *See:*
  Florian Klück, Martin Zimmermann, Franz Wotawa, and Mihai Nica. ***Genetic algorithm-based test parameter optimization for ADAS system testing****. In Proceedings of the 19th IEEE International Conference on Software Quality, Reliability, and Security (QRS), 2019.*



71

Another example (cont.)

- Use Time-to-Collision (TTC) as fitness function
- Parameters:

Parameter (Ego)	Unit	Parameter (GVT)	Unit
EgoTarget	$[m/s]$	GvtTarget	$[m/s]$
EgoSpeedChange	$[m/s^2]$	GvtSpeedChange	$[m/s^2]$
EgoInitSpeed	$[m/s]$	GvtInitSpeed	$[m/s]$
EgoOffset	$[m]$	GvtYPosition	$[m]$

- Results:

Testing Technique	Number of Tests	min. TCC Test Set 1	min. TTC Test Set 2	min. TTC σ
Random	106	0.270000	1.280000	-
Genetic	106	0.347000	0.372000	0.0176
Random	1,065	0.270000	0.310000	-
Genetic	1,060	0.271359	0.271372	$9.19{\cdot}10^{-6}$

72

Another example (cont.)

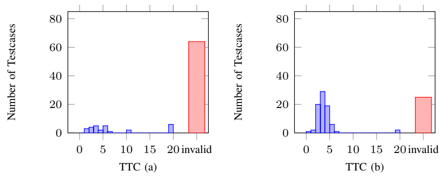


Figure 3. Histogram depicting the resulting TTC distribution of low budget test sets for (a) random testing (one test set) and (b) genetic algorithm tests (one test set).

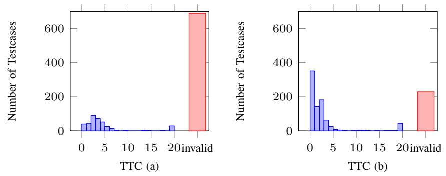


Figure 4. Histogram depicting the resulting TTC distribution of high budget test sets for (a) random testing (one test set) and (b) genetic algorithm tests (one test set).

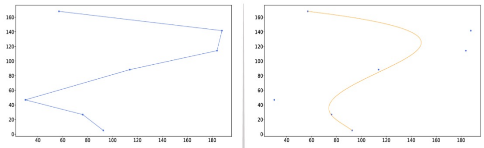
- Random testing performs well
  - The probability of finding a fault is high
- GA's focus a lot towards finding optimal test cases
  - Lower number of invalid tests

73

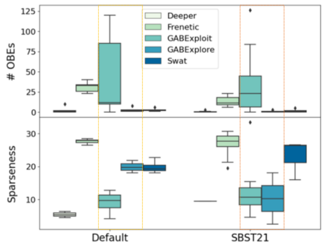
Does SBT also work for testing different driving functions in varying test environments?

[4] SBST Tool Competition 2021

- In [4] we participated in in the first Cyber-Physical Systems Testing Tool Competition and contributed a test case generator for automatic road generation for testing a lane-keeping system (ALKS) in BeamNG.
- Test case generator:
  - GA that modifies for control point arrangements of a Bezier curves to result in challenging road geometries
  - The test generator can be applied in two different search variants:
    - Exploitative search (GABExploit)
    - Exploratory search (GABExplore)
- SBST Tool Competition Outcome:
  - GABExploit generated a substantial amount of OEBs
  - GABExplore only performed well on higher time budgets



Name	Map Size	Speed Limit	Gen. Budget	Tolerance
DEFAULT	200m × 200m	None	5 h	0.95
SBST21	200m × 200m	70 Km/h	2 h	0.85



74/25

[4] Sebastiano Panichella, Alessio Gambi, Fiorella Zampetti, and Vincenzo Riccio. Sbst tool competition 2021. In 2021 IEEE/ACM 14th International Workshop on Search-Based Software Testing (SBST), pages 20–27, May 2021.

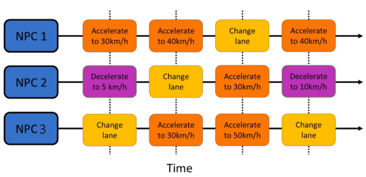
74

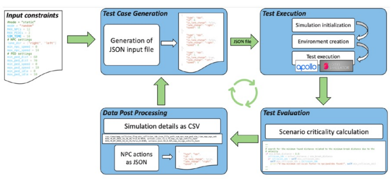
Does SBT also work for testing different driving functions in varying test environments?

[5] 2021 IEEE Autonomous Driving AI Test Challenge

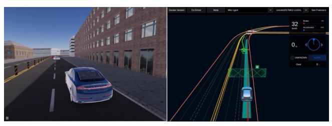
- In [5] we participated in 2021 IEEE Autonomous Driving AI Test Challenge, we contributed an approach for the automated generation of diverse critical scenarios for testing the Apollo AD stack in LG SV

- Approach:
  - GA-based behavior sequence optimization
  - The approach generates and modifies lists of actions, which are executed by the NPCs during the simulation
  - Open search space, not restricted to underlying scenario parameters

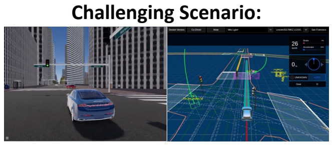




Critical Scenario:



Challenging Scenario:



- Conclusion:
  - Q3: Yes - The SBT approach can be transferred for testing different driving functions in varying test environments
    - SBT revealed ALKS failure in BeamNG environment
    - SBT revealed failure in the Apollo full driving stack in LG SVL simulator

75/25

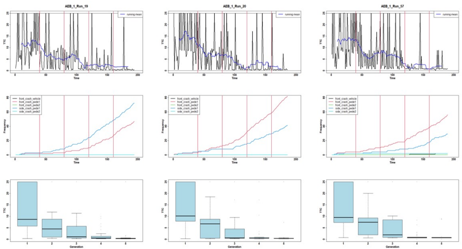
[5] D. Kaufmann, L. Klampfl, F. Kluck, M. Zimmermann, and J. Tao. Critical and challenging scenario generation based on automatic action behavior sequence optimization: 2021 ieee autonomous driving ai test challenge group 108. In 2021 IEEE International Conference On Artificial Intelligence Testing (AITest), pages 118–127, Los Alamitos, CA,USA, aug 2021. IEEE Computer Society.

75

Can we provide guarantees and determine when to stop testing?

Final considerations

- Software and system testing can never prove the absence of failure…
  - If SBT generates just one failing test case, we know the system is faulty!
  - If SBT generates no failure -> when to stop SBT?
- Theoretical perspective:
  - No hard guarantees for SBT
    - No guarantee all critical regions covered
    - Genetic operators and cost-function might bias the search
- Practical perspective:
  - Advantage: SBT identifies failures fast
  - High number of failing test cases in all our ADAS studies
    - Observation: Cost-function reliably converged towards critical regions between generations
  - With increasing maturity, the number of failures in the system should also decrease.
  - Given the cost function considers parameters related to failure:
    - No convergence between generations -> SBT has lost its advantage to identify failures fast.
    - Final assessment using other testing strategies (e.g., a high strength combinatorial test suite)



76/25

76

Further reading

(1) *Hermann Felbinger and Florian Klück and Yihao Li and Mihai Nica and Jianbo Tao and Franz Wotawa and Martin Zimmermann.* ***Comparing two systematic approaches for testing automated driving functions***. *In Proceedings of the 8th IEEE International Conference on Connected Vehicles and Expo (ICCVE), Graz, Austria, 2019.*

(2) *Florian Klück, Franz Wotawa, Martin Zimmermann and Mihai Nica.* ***Performance comparison of two search-based testing strategies for ADAS System Validation***. *In Proceedings of the 31st IFIP International Conference on Testing Software and Systems (ICTSS), Paris, France, 2019.*

(3) *Florian Klück, Martin Zimmermann, Franz Wotawa, and Mihai Nica.* ***Genetic algorithm-based test parameter optimization for ADAS system testing***. *In Proceedings of the 19th IEEE International Conference on Software Quality, Reliability, and Security (QRS), 2019.*

77

Summary SBT

- Genetic programming and GA have been used for test suite generation successfully (not only for AD/ADAS, e.g., EvoSuite for Java)
- Require a mapping of test generation into the search paradigm, e.g. the GA
- Can also be used to test safety-critical functions (e.g., in the case of automated automotive functions)
- Sometimes random testing seems to be sufficient as well!

78

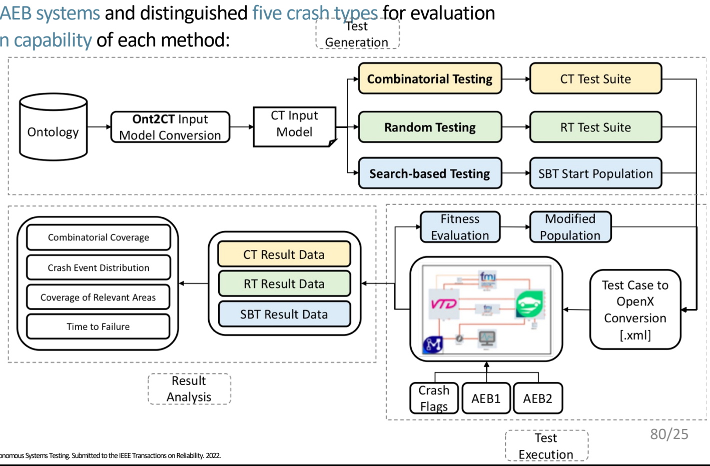
Comparing OBT & SBT

79

How does SBT perform compared to combinatorial testing?

[3] An Empirical Comparison of Combinatorial Testing and Search-based Testing in the context of Automated and Autonomous Systems Testing

- In [3] we performed an empirical comparison of CT and SBT in the context of ADAS testing (RT as baseline)
- 3rd AEB case study:
  - **Test Generation:** all methods share the same CT input model to generate test suits of similar size
  - **Test Execution:** we performed our study on two AEB systems and distinguished five crash types for evaluation
  - **Result Analysis:** we focus on the failure detection capability of each method:
    - Distribution of AEB1 and AEB2 crash events
    - Coverage (t-way combinatorial coverage)
    - (Time-to-failure)
    - (Similarity between critical scenarios)



Crash flags	Description
FCV	Collision with front vehicle
FCP1	Front Collision with pedestrian 1
FCP2	Front Collision with pedestrian 2
SCP1	Side Collision with pedestrian 1
SCP2	Side Collision with pedestrian 2

[3] Florian Klück, Yihao Li, Jianbo Tao, and Franz Wotawa. An Empirical Comparison of Combinatorial Testing and Search-based Testing in the context of Automated and Autonomous Systems Testing. Submitted to the IEEE Transactions on Reliability. 2022.

80

How does SBT perform compared to combinatorial testing?

[3] Test Case Generation:

- CT-based:
  - CT test suite generation using AVL "Load Matrix for Software":
    - CT2: 978 test cases
    - CT3: 21,418 test cases
- RT-based:
  - Uniform distribution:
    - First, select parameters from CT input model at random
    - Second, remove forbidden combinations
    - Repeat until RT test suite same size as CT2 i.e., 978 test cases
- SBT-based:
  - GA optimization problem:
    - Genes: scenario players
    - Chromosomes: scenario player properties
    - Scenario type: random during seed generation
    - Constraints checked on every individual (seed, crossing, mutation)
  - We aim for a SBT test suite size similar to CT2:
    - SBT test run: populate 40 individuals over 5 generations
    - SBT test suite: combine 6 independent test runs

CT Input Model

```
Node: DynamicPart
Parameter: NumberOfVehiclePlayer; (enum) : 1, 2, 3
Parameter: NumberOfPede; (enum) : 1, 2, null

Node: EgoVehicle
Parameter: Start_speed; (enum) : 0,5,10,15,20,25,30,35,40,45,50,55,60,65,70,75,80,85,90,95,100,105,110,115,120,125,130
Parameter: Direction; (enum) : 0.30.60.90.120.150.180.210,240,270,300,330
Parameter: VehicleType; (enum): model1,model2,model3,model4,model5,model6,model7,model8
Parameter: Offset_s; (enum):  -50,-25,0, 25, 50
Parameter: Offset_t; (enum):  0
Parameter: Rate; (enum) :  0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Parameter: Target_Speed; (enum) : 0,5,10,15,20,25,30,35,40,45,50,55,60,65,70,75,80,85,90,95,100,105,110,115,120,125,130

Node: Vehicles_Players
Parameter: Start_speed; (enum): 0,10,20,30,40,50,60,70,80, null
Parameter: Direction; (enum): 0.30.60.90.120.150.180.210,240,270,300,330, null
Parameter: VehicleType; (enum): model1,model2,model3,model4,model5,model6,model7,model8, null
Parameter: Offset_s; (enum): 0,4.5,null
Parameter: Offset_t; (enum): 10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40 null
Parameter: Rate; (enum) :  0, 1, 2, 3, 4, 5, 6, 7, 8,10, null
Parameter: Target_Speed; (enum) : 0,10,20,30,40,50,60,70,80, null
Parameter: DriverType; (enum) : No Driver, DefaultDriver, null

Node: Pedestrains_Objects
Parameter: Start_speed; (enum) : 0, 2.8, 5.6, 8.3, 11.1, 13.9, 16.7, 19.4, 22.2, 25, 27.8, 30.6, 33.3, 36.1, null
Parameter: Direction; (enum) : 0.30.60.90.120.150.180.210,240,270,300,330, null
Parameter: Offset_s; (enum): -9,-10,-11,-12,-13,-14,-15,-16,-17,-18,-19,null
Parameter: Offset_t; (enum): 10,11,12,13,14,15,16,17,18,19,20, null
Parameter: PedeType; (enum): Adult, Child, Wheelchair, Animal, Ballon, Paper, Dog, Stone, Adult_w_Bicycle, Adult_w_child, custom1, custom2, custom3, null
Parameter: Rate; (enum) :  0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, null
Parameter: Target_Speed; (enum) : 0, 2, ,4,6,8,10,12,14,16,18,20, null
```

GA Representation

Individual Player	Gene 0 Ego	Gene 1 Vehicle I	Gene 2 Vehicle II	Gene 3 Vehicle III	Gene 4 Pedestrian I	Gene 5 Pedestrian II	Scenario Type
Indv 0	E 0	VI 0	VII 0	VIII 0	PI 0	PII 0	E-3V-2P
Indv 1	E 1	VI 1	VII 1	null	PI 1	null	E-2V-1P
Indv 2	E 2	VI 2	VII 2	VIII 2	null	null	E-3V-0P
Indv n	E n	VI n	null	null	null	null	E-1V-0P

Gene Indv 1	Chrom 0 start speed	Chrom 1 target speed	Chrom 2 offset s	Chrom 3 offset t	Chrom 4 rate	Chrom 5 Player	Chrom 6 driver type
E 1	11.11	28	0	0	5	v-model 1	n/a
VI 1	10	2	0	120	6	v-model 2	default driver
VII 1	0	0	4.5	175	0	v-model 1	no driver
VIII 1	null	null	null	null	null	null	null
PI 1	0.56	n/a	13	200	3	p-model	n/a
PII 1	null	n/a	null	null	null	null	null

[3] Florian Klück, Yihao Li, Jianbo Tao, and Franz Wotawa. An Empirical Comparison of Combinatorial Testing and Search-based Testing in the context of Automated and Autonomous Systems Testing. Submitted to the IEEE Transactions on Reliability. 2022.

81

How does SBT perform compared to combinatorial testing ?

QAMCAS

[3] Result Analysis:

AEB1 Crash Event Distribution

Table 1.5: SBT, RT and CT test generation and crash summary for AEB1

SBT	AEB1	Total	Pass	Fail	FCV	FCP1	SCP1	FCP2	SCP2
1	SBT01	1024	806	218	0	162	56	0	1
2	SBT02	983	857	126	0	31	94	1	0
3	SBT03	1002	802	200	0	120	79	0	1
4	SBT04	1010	743	267	0	216	50	0	1
5	SBT05	1005	807	198	3	151	41	4	1
6	SBT06	988	884	104	0	43	55	4	3
7	SBT07	971	743	228	0	203	24	0	5
8	SBT08	990	771	219	0	156	61	0	2
9	SBT09	963	764	199	0	165	34	0	0
10	SBT10	1021	801	220	2	134	82	0	2
RT	AEB1	Total	Pass	Fail	FCV	FCP1	SCP1	FCP2	SCP2
1	RT01	978	949	29	0	19	9	1	0
2	RT02	978	936	42	0	24	17	1	0
3	RT03	978	943	35	1	16	17	1	0
4	RT04	978	938	40	0	24	16	0	0
5	RT05	978	944	34	0	17	16	0	1
6	RT06	978	926	52	0	34	18	0	0
7	RT07	978	949	29	0	12	17	0	0
8	RT08	978	939	39	0	19	19	0	1
9	RT09	978	932	46	0	24	20	2	0
10	RT10	978	937	41	0	21	17	3	0
CT2	AEB1	Total	Pass	Fail	FCV	FCP1	SCP1	FCP2	SCP2
1	CT2_01	978	942	36	0	12	18	4	2
CT3	AEB1	Total	Pass	Fail	FCV	FCP1	SCP1	FCP2	SCP2
1	CT3_01	21418	19567	1851	2	604	999	159	87

- FCV is the most difficult crash type to detect:
  - SBT generated five FCV scenarios in two test runs (SBT05 and SBT10)
  - RT detected one FCV scenario in test run (RT03)
  - CT2 triggered no FCV crash event
  - CT3 detected two FCV crash events
- All remaining crash events got detected by each method (some less frequent)
- AEB1 has noticeable defects in avoiding collision with pedestrian 1

AEB1 Crash Event Probability Analysis

AEB1	P(Fail)	P(FCV)	P(FCP1)	P(SCP1)	P(FCP2)	P(SCP2)
$\overline{SBT}$	19.85%	0.05%	13.86%	5.77%	0.09%	0.16%
$\overline{RT}$	3.71%	0.01%	1.99%	1.61%	0.08%	0.02%
CT2_01	3.68%	0.00%	1.23%	1.84%	0.41%	0.20%
CT3_01	8.64%	0.01%	2.82%	4.66%	0.74%	0.41%

$$P(Fail) = \frac{Fail}{Total} * 100[\%]$$

$$P(FCV) = \frac{FCV}{Total} * 100[\%]$$

- SBT scenarios show higher probability to detect failure (19.85%) compared to RT, CT2, and CT3, especially for FCV (0.05%), FCP1 (13.86%), and SCP1 (5.77%)
- CT2 and CT3 scenarios show higher probability to detect FCP2 (0.41%) and SCP2 (0.20%) compared to SBT and RT

82/25

[3] Florian Klück, Yihao Li, Jianbo Tao, and Franz Wotawa. An Empirical Comparison of Combinatorial Testing and Search-based Testing in the context of Automated and Autonomous Systems Testing. Submitted to the IEEE Transactions on Reliability. 2022.

82

How does SBT perform compared to combinatorial testing ?

QAMCAS

[3] Result Analysis:

AEB2 Crash Event Distribution

Table 1.7: SBT, RT and CT test generation and crash Summary AEB 2

SBT	AEB2	Total	Pass	Fail	FCV	FCP1	SCP1	FCP2	SCP2
1	SBT01	976	867	109	96	3	11	0	0
2	SBT02	916	287	629	581	37	45	0	0
3	SBT03	897	309	588	537	51	65	0	17
4	SBT04	990	746	244	218	20	51	0	0
5	SBT05	923	285	638	581	68	55	0	1
6	SBT06	931	282	649	602	84	32	0	0
7	SBT07	925	264	661	589	69	28	0	16
8	SBT08	907	677	230	188	11	33	0	0
9	SBT09	1006	896	110	69	2	35	0	8
10	SBT10	946	837	109	98	2	10	0	0
RT	AEB2	Total	Pass	Fail	FCV	FCP1	SCP1	FCP2	SCP2
1	RT01	978	960	18	10	6	2	0	0
2	RT02	978	953	25	17	5	3	0	0
3	RT03	978	952	26	18	6	2	0	0
4	RT04	978	937	41	27	9	5	0	0
5	RT05	978	936	42	16	5	20	0	1
6	RT06	978	947	31	14	13	4	0	0
7	RT07	978	945	33	20	3	10	0	0
8	RT08	978	929	49	16	5	28	0	0
9	RT09	978	942	36	17	2	17	0	0
10	RT10	978	937	41	23	1	17	0	0
CT2	AEB2	Total	Pass	Fail	FCV	FCP1	SCP1	FCP2	SCP2
1	CT2_01	978	901	77	57	3	12	1	4
CT3	AEB2	Total	Pass	Fail	FCV	FCP1	SCP1	FCP2	SCP2
1	CT3_01	21418	19520	1898	1356	122	96	286	38

- FCP2 is the most difficult crash type to detect, followed by SCP2:
  - SBT and RT both fail to generate a FCP2 scenarios
  - SBT shows slight advantages over RT in detecting SCP2
  - CT2 and CT3 are both able to detect all crash types
- AEB2 seems more vulnerable with noticeable defects that lead FCV and FCP1

AEB2 Crash Event Probability Analysis

AEB2	P(Fail)	P(FCV)	P(FCP1)	P(SCP1)	P(FCP2)	P(SCP2)
$\overline{SBT}$	42.81%	38.44%	3.75%	3.90%	0.00%	0.49%
$\overline{RT}$	3.30%	1.62%	0.75%	0.88%	0.00%	0.01%
CT2_01	7.87%	5.83%	0.31%	1.23%	0.10%	0.41%
CT3_01	8.86%	6.33%	0.57%	0.45%	1.34%	0.18%

- SBT scenarios always show higher probability to detect failure, except for FCP2
- CT3 and CT2 highest and second highest possibility to detect FCP2

83/25

[3] Florian Klück, Yihao Li, Jianbo Tao, and Franz Wotawa. An Empirical Comparison of Combinatorial Testing and Search-based Testing in the context of Automated and Autonomous Systems Testing. Submitted to the IEEE Transactions on Reliability. 2022.

83

How does SBT perform compared to combinatorial testing ?

QAMCAS

[3] Result Analysis:

T-way Combinatorial Coverage Analysis

SUT	Method	2-way coverage	3-way coverage	4-way coverage	5-way coverage
AEB1	$\overline{SBT}$	75,7%	30,2%	8,9%	2%
AEB2	$\overline{SBT}$	74,5%	29,2%	8,8%	2%
AEB1\AEB2	$\overline{RT}$	79,9%	34,1%	10,8%	3%
AEB1\AEB2	CT2_01	100%	42%	13%	4%

- SBT has a slightly lower coverage than RT with respect to the t-way coverage
- Each CT2 coverage is higher than the corresponding n-way (n=2,3,4,5) coverage for SBT and RT

- Conclusion:
  - Q2: SBT identifies system failures faster than CT and RT. However, CT shows more reliable coverage of failure types
    - CT and SBT tend to be complementary to each other.
      - On the one hand, CT is more effective than SBT in terms of the detected crash types
      - On the other hand, SBT is likely to be more efficient than CT having a higher crash detection probability
    - RT does not appear to have a clear advantage over CT or SBT

84/25

[3] Florian Klück, Yihao Li, Jianbo Tao, and Franz Wotawa. An Empirical Comparison of Combinatorial Testing and Search-based Testing in the context of Automated and Autonomous Systems Testing. Submitted to the IEEE Transactions on Reliability. 2022.

84

Summary and conclusions

86

Summary

- Both SBA and OBT/CT work fine for detecting faults of AD/ADAS functionality
  - AEB, ALKS, AD (Apollo)
- OBT/CT comes with guarantees:
  - Combinatorial strength
  - Ontology coverage
- There are no guarantees in the case of SBT
- Random Testing is often also working very well

87

Research questions

- When do use OBT/CT and when SBT?
  - Come up with guidelines for testing methods for AD/ADAS
- Oracle problem is not always that easy to solve!
  - TTC is not always the best measure
  - There might be other properties to be checked during testing
- OBT/CT comes with a high computational footprint
  - Improve using OBT/CT
- Other application areas for OBT

88

Testing is not enough!

- Even after rigorous testing we may face trouble
  - Search space for testing is huge!
  - n concepts and k values: k^n
  - ***Example**: 100 concepts, each domain of size 5 leads to 10^{70} different tests*
- Need to evaluate driving behavior during operation!
- Fail-operationality required too!

Anomaly Detection Through Explanations

by

Leilani Hendrina Gilpin

B.S., University of California, San Diego (2011)
M.S., Stanford University (2013)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
September 2020

© Massachusetts Institute of Technology 2020. All rights reserved.

Author ....
Department of Electrical Engineering and Computer Science
August 28, 2020

Certified by ....
Gerald Jay Sussman
Panasonic Professor of Electrical Engineering
Thesis Supervisor

Certified by ....
Lalana Kagal
Principal Research Scientist in CSAIL
Thesis Supervisor

Accepted by ....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

89

QAMCAS

Thank you for your attention!

Questions?

Christian Doppler Forschungsgesellschaft

Federal Ministry for Digital and Economic Affairs

AVL

TU Graz

90