


# Challenging Autonomy with Combinatorial Testing

Simon Diemert   
Critical Systems Labs Inc.  
Vancouver, Canada  
simon.diemert@cslabs.com

Adam Casey  
Critical Systems Labs Inc.  
Vancouver, Canada  
adam.casey@cslabs.com

Jeremiah Robertson  
Motional AD Inc.  
Boston, USA  
jeremiah.robertson@motional.com

**Abstract**—This paper describes an input space modelling and test generation method called CACTus (Challenging Autonomy with Combinatorial Testing) that creates a suite of ‘challenge scenarios’ for autonomous systems. Though the parameter space for autonomous systems is vast, CACTus helps to reduce the parameter space using combinatorial testing and by incorporating expert judgement into the formulation of the scenarios. The resulting scenarios can be executed on appropriate test infrastructure, such as simulators or hardware-in-loop testing. CACTus may be used by practitioners to exercise systems as part of efforts to obtain compliance with standards like ISO 21448 or UL 4600. The method is applied to generate test scenarios for the perception system of a commercial autonomous vehicle.

**Index Terms**—combinatorial testing, input modeling, autonomous systems, autonomous vehicles

## I. INTRODUCTION

Autonomous vehicles (AVs), such as Motional’s commercial robotaxi, must handle a diverse range of external conditions, including weather, varying road geometries, agent appearances, and agent behaviours. Requirements to manage external conditions are addressed by various industrial standards, such as ISO 21448 - *Road vehicles - Safety of the intended function* [1] and UL 4600 - *Evaluation of Autonomous Products* [2]. The number of combinations of external conditions is vast. Consider just the variability associated with a single pedestrian; attributes such as height, age, hair style, hair color, skin colour, clothing colour, clothing style, body position, and so forth might impact the performance of an AV’s perception sub-system. Machine learning (ML) methods such as Convolutional Neural Networks (CNNs), which are widely used to perform the object detection and classification tasks within perception sub-systems, can be sensitive to minor variation in the inputs, especially when the inputs are ‘novel’, i.e., where not adequately represented in the ML component’s training data [3]. Failure to detect or correctly classify objects has contributed to fatal accidents involving AVs [4], [5].

Safety-critical systems typically undergo extensive verification and validation, including testing, to demonstrate they have safely implemented the desired behaviour. For instance, both ISO 21448 and ISO 26262 - *Road vehicles - Functional safety* [6] provide guidance on engineering practices for developing safe and dependable automotive systems; testing is a cornerstone method in both standards. However, testing of AVs poses a significant challenge: it is simply not possible to test (in simulation or the field) every possible input combination to an AV’s perception, prediction, and planning pipeline [7], [8].

Even if testing every input (or significant subset) was feasible, it is likely that many tests would be uninformative (i.e., not revealing defects or safety issues) resulting in an ineffective use of resources. Given finite testing resources, there is a need to select test cases that have a higher likelihood of finding defects or limitations of the AV’s system(s).

Moreover, simply testing the system(s) is not sufficient. It is also necessary to assess the quality (coverage, completeness, etc.) of the test suite to demonstrate that it has covered a breadth of input conditions. Metrics such as “number of miles driven” are only meaningful when paired with information about the breadth of conditions encountered [7], [9]. For conventional (‘if-then-else’ logic) software and electronics, standards such as ISO 26262 recommend using methods like equivalence classes, structural coverage measures (e.g., lines of code exercised), and modified condition/decision coverage (MC/DC) measures to assess test suite completeness. However, these methods are not effective for ML-enabled systems, either due to fundamental differences between conventional (‘if-then-else’ logic) software and ML algorithms or simply a failure to scale to the massive input spaces often used for ML-enabled systems. For instance, some researchers have adapted structural coverage measures to neural networks by measuring the proportion of neurons activated [10]–[12]; however, these measures might not be an effective measure of test quality [13]–[15]. Combinatorial Testing (CT) is a method that has shown promise for conventional and ML-enabled systems, both for generating test cases and establishing completeness (up to some combinatorial strength) of a test suite [16]–[19].

### A. Combinatorial Testing

CT is a method to generate test sets covering the possible combinations of input variables to a desired strength  $t$ . The strength  $t$  defines the number of different parameters that will be tested for interaction, with a higher value of  $t$  resulting in a greater number of generated tests, and ideally greater confidence in the thoroughness of testing. Consider a system with  $k$  parameters  $p_1, \dots, p_k$  having  $m_1, \dots, m_k$  values each. For the trivial case where  $t = 1$ , combinatorial testing would produce a test set where every value of each parameter is present at least once, producing at least as many test cases as the greatest number of values associated with a single parameter,  $m_{max}$ . This minimal case fully covers the possible individual inputs to the system, but does not investigate the possible interactions between inputs. At the other end, using

$t = k$  would test every possible combination of input values and produce test cases numbering  $\prod_{i=1}^k m_i$ . Values of  $t$  between 1 and  $k$  will produce test sets numbering between these extremes. It has been suggested that  $t \leq 6$  is sufficient for most conventional software systems [20].

ISO 21448 (Sections 10.3 and 10.4) identifies CT as a method for deriving test cases over a range of environmental conditions and says that “judicious use of the principles of combinatorial testing can be applied” [1]. However, it does not provide further guidance as to how this method should be used to create meaningful test cases, especially for large or complex operational domains, such as those encountered for an AV. This poses a problem for practitioners who want to apply the guidance from industrial standards but become stuck when those methods do not scale to the vast input spaces for applications like autonomous driving.

### B. Overview of Contribution

There is a need to create meaningful test suites that exercise autonomous systems in a manner that is consistent with the guidance from industrial standards, covers a variety of input combinations, and that are practical in terms of scale. In this paper, we introduce CACTus (Challenging Autonomy with Combinatorial Testing), a method for modelling the system input space and deriving a suite of ‘challenge scenarios’ that cover (up to a desired combinatorial strength) relevant aspects of an AV’s operating conditions. The results of applying CACTus can be used as an input to field or simulation-based test activities. More concretely, the contribution of this paper is two-fold: 1) a description of the CACTus method for generating challenge scenarios for autonomous systems; and 2) an experience report of applying CACTus to generate test scenarios for Motional’s AV’s perception system.

## II. THE CACTUS METHOD

CACTus (Challenging Autonomy with Combinatorial Testing) is an input space modelling and test scenario generation method. The output of CACTus is a set of concrete test scenarios that exercise the system-under-test over the operational design domain (ODD). The scenarios output by CACTus could be used for a range of testing activities, including bench testing, software-in-loop simulation, hardware-in-loop simulation, or closed course testing. CACTus can also be used “in reverse” to assess the combinatorial coverage of a pre-existing set of concrete scenarios; see further discussion in Section V. An overview of CACTus is depicted in Figure 1.

In this section, CACTus is illustrated using the example of an AV’s perception system that detects and classifies pedestrians, cyclists, and vehicles. The perception system consumes raw sensor data from lidar, radar, and cameras, and outputs object detections that are used by other systems to control the vehicle’s motion. Failing to detect an object could result in the vehicle executing a trajectory that collides with the missed object. Therefore, it is important that the perception system accurately detects objects, especially vulnerable road users like pedestrians or cyclists.

### A. Step 1) Define Operational Design Domain (ODD)

The first step of CACTus is to define the ODD for the system-under-test. The ODD should describe the environment and situations that the system is expected to encounter during its operations [2]. For an AV, an ODD should include aspects such as environmental conditions (e.g., precipitation, time of day), geographic regions, road geometries (e.g., lane size, lane delineation, road curvatures), traffic signals and signs, and reasonably foreseeable agent types, appearances, and behaviors. Several academic and industrial resources address ODD definition in more detail, and we refer interested readers to this work [2], [21]–[24]. For CACTus, it is important that the ODD be expressed using a taxonomy (or similar structure) to categorize different aspects of the ODD. This is necessary for subsequent steps that (programmatically) use the ODD to generate scenarios. Additionally, it is likely that some aspects of the ODD will not be compatible with each other. For example, an object of type ‘vehicle’ does not have a ‘hair length’ attribute but an object of type pedestrian does. Similarly, some environmental conditions are incompatible such as heavy rain and clear skies. Therefore, in addition to an ODD taxonomy, the ODD specification should also include compatibility matrices describing which aspects of the ODD may be reasonably combined when creating scenarios. As with the ODD itself, these matrices are represented in a format that can be easily consumed by a computer program.

a) *Example - ODD for Object Perception System:* For our illustrative example, we consider the ODD for a perception system designed to detect other road users. For this ODD, we considered environmental conditions (e.g., amount of rainfall, dust, ambient lighting, and temperature, etc.), sub-types of each type of road users (e.g., vehicle has sub-types SUV, sedan, pickup truck, box truck, bus, etc.), and attributes of each type (e.g., pedestrians vary by height, aspect ratio, skin color, clothing style, etc.). Each element of the ODD has a unique identifier that also defines its location within the taxonomy, e.g., 1.1.1.1.3.2 is a livestock trailer. In total, the ODD for our perception system contains 199 object types, 57 environmental conditions, and 166 attributes. We have defined two compatibility matrices, one for object attributes and another for environmental conditions.

### B. Step 2) Identify Logical Scenario(s)

In the second step of the CACTus methodology, expert knowledge and intuition is combined with technical data (e.g., system architecture, ODD, SOTIF analysis results, etc.) to identify one or more logical scenarios. A logical scenario is an abstract description of a family of tests that will exercise a particular behavior of the system-under-test. Logical scenarios cannot be directly executed, instead they must be instantiated into one or more concrete scenarios that can be executed on the corresponding test infrastructure (e.g., within a simulator); see Step 4 below.

The description of each logical scenario must include a *raison d’être*, i.e., it must capture a credible set of conditions that

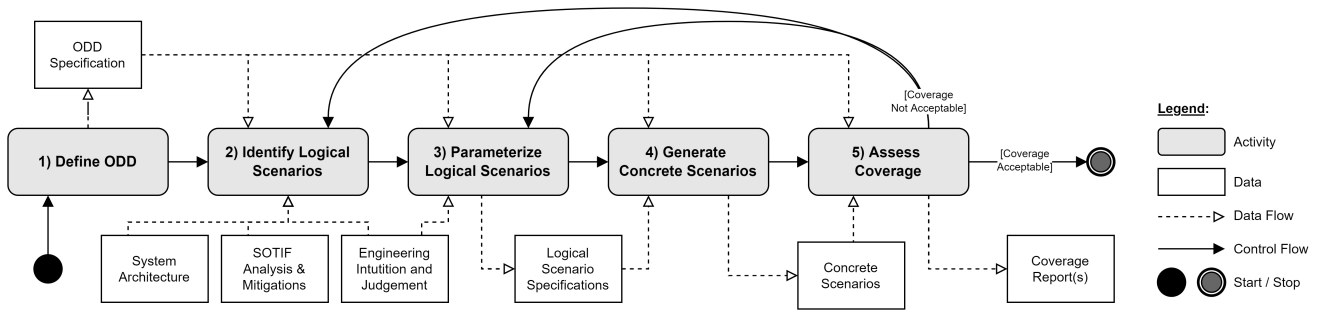


Fig. 1. Overview of CACTus method for generating concrete test scenarios.

are expected to challenge and reveal limitations of the system-under-test. In scientific terms, the description of a logical scenario should contain a challenge hypothesis that describes a belief about the conditions that will be difficult for the system-under-test to handle. The goal of CACTus is to generate test cases that demonstrate that the challenge hypothesis is true (i.e., that the system-under-test is indeed challenged by the conditions). Execution of these tests generates evidence that can be used to determine if the challenge hypothesis can be rejected, in support of an alternative hypothesis that the system-under-test is not challenged by the identified conditions. Additionally, part of specifying a challenge hypothesis is describing the hypothesis rejection criteria. After executing the tests, the results should be assessed against the criteria to determine if the challenge hypothesis can be rejected. A simplistic rejection criterion might be that all tests pass (according to some pre-defined pass criteria). However, other rejection criteria are also possible. Our notion of a challenge hypothesis parallels that of a null hypothesis used in empirical studies; from this perspective, there are a few important points.

First, like a null hypothesis, rejecting a challenge hypothesis does not unquestionably prove the validity of the alternative hypothesis. Though it is common in hypothesis testing to accept the alternative hypothesis if the null hypothesis is rejected, this does not mean that the alternative hypothesis is absolutely, unquestionably true. Replication experiments or different experimental conditions might yield different results. If the challenge hypothesis is rejected, one cannot assume that the system-under-test has perfect performance in challenging conditions. Indeed, when testing complex systems, such as AVs, it is not possible to test every permutation of inputs such that claims of “perfect performance” are proven with certainty.

Second, the logical scenario’s formulation (and subsequent detailed specification, see Step 3) must be viewed through the lens of demonstrating the validity of the challenge hypothesis. That is, the analyst specifying the logical scenario must adopt an adversarial mindset to challenge the system-under-test in accordance with the challenge hypothesis. Without this mindset, rejection of the challenge hypothesis cannot be used to support a conclusion about adequacy of the system-under-test. The confidence that can be derived from rejecting the hypothesis that the system is challenged by certain conditions

is only as strong as the challenge that is presented by those conditions. If the analyst were to specify unchallenging logical scenarios, rejecting the challenge hypothesis, and inferring that the system is not challenged by these conditions, would provide little confidence in the system’s real-world performance.

Finally, in empirical research it is common to use statistical tests (e.g., a T-test) to determine whether to reject a null hypothesis. One advantage of statistical testing is that probabilities of Type I and II errors can be expressed, which provide a measure of confidence in the results. Though CACTus does not exclude the possibility of statistical testing, the rejection criteria are not limited to statistical tests. Non-statistical criteria (such as number of tests passed) is suitable for our purpose, but this comes at the cost of not being able to measure the probabilities of Type I and II errors.

As a practical matter, when describing a logical scenario, in addition to a challenge hypothesis and assessment criteria, we have also found it helpful to include: 1) a graphical representation of the scenario (typically one instance of a concrete scenario, for illustrative purposes); 2) a narrative description of the scenario timeline/events; 3) traceability links to relevant system-level hazards, failure modes, or SOTIF environmental triggers; and 4) traceability links to applicable system requirements, including particular SOTIF mitigations.

#### a) Example - Logical Scenario for Pedestrian Variant:

For our illustrative example, we consider a logical scenario with a single unoccluded pedestrian positioned near the ego vehicle; the pedestrian is one of many possible variants. In this scenario, there are no additional objects, and the environmental conditions are “fair” (i.e., daytime, no precipitation, dust, or fog, etc.). The purpose of the scenario is to test whether the vehicle’s perception system accurately detects and classifies variants of pedestrians; to this end we formulate the following challenge hypothesis:  $CH_0$  - *The AV’s perception system does not detect pedestrians near the ego vehicle with novel appearances or behaviors.*

In this context, the term “novel” is borrowed from the ML research community and refers to combinations of object attributes did not appear (with adequate frequency) in the data used to train the object detection or classification algorithm(s). This challenge hypothesis is based on the widely reported limitation of ML-based systems at recognizing novel or out-of-distribution inputs.

To reject the challenge hypothesis  $CH_0$ , we require that all pedestrian variants near the ego vehicle are detected and correctly classified. That is, in all concrete scenarios generated from this logical scenario, the single pedestrian is detected and correctly classified. Note that this example is limited to the task of perceiving objects near the vehicle, so the rejection criteria does not consider whether the vehicle’s motion control system(s) avoid the pedestrian; though that is certainly important, it is not within the scope of this example.

As noted above, we have found that graphical representations of the scenario help to communicate the scenario intent. An instance (one possible concrete scenario) of this logical scenario is depicted in Figure 2, where a pedestrian wearing an unusual hat that resembles a traffic cone and carrying an object is shown crossing the road in front of the ego vehicle.



Fig. 2. Depiction of pedestrian variant challenge scenario; ego vehicle in yellow, pedestrian variant has an unusual hat.

### C. Step 3) Parameterize Logical Scenarios

Once the logical scenarios are identified, their definitions should be refined to include relevant parameters and the corresponding range of values for each parameter. For example, the distance between the ego vehicle and an object of interest in the scenario could be a parameter that takes values in the range 0 and 100 meters. The output of this parameterization step is a set of scenario specifications (expressed in a machine-readable format, such as the JavaScript Object Notation (JSON)) which are used to generate concrete scenarios in Step 4.

Analysts parameterizing scenarios should make use of expert knowledge and judgement to select parameters (and values) that align with the logical scenario’s intent, as expressed by the challenge hypothesis. When parameterizing scenarios, it might be tempting for analysts to pick many parameters and values that cover every conceivable aspect of the scenario. For example, in the pedestrian variant example (see Figure 2), the number of lanes of traffic could be identified as a parameter ranging from one through eight. This might increase confidence in the completeness of the logical scenario and the generalizability of the results. However, it also increases the number concrete scenarios generated in Step 4 and might contribute to scalability problems without adding much value in terms of challenging the system-under-test. Instead, when parameterizing scenarios it is important for experts, guided by the challenge hypothesis, to use their judgement and select the most relevant parameters.

One way to reduce the parameter space for the scenario is to use equivalence classes for parameter values, rather than complete numerical ranges. For example, if the ego vehicle’s speed is a parameter, then for scenarios that are not particularly sensitive to vehicle speed, equivalence classes like 0, 1, 5, 10, 15, 20, ... m/s might be appropriate. Again, engineering judgement must be used when selecting equivalence classes. Our example examines a perception system that is not expected to be sensitive to speed; but a motion-planning or emergency-braking system might be much more sensitive to speed, so the input partitioning would require greater attention.

Another way to reduce the parameter space is to define constraints between the values of parameters to avoid invalid or meaningless parameter combinations. For example, if an object’s speed and object type (pedestrian, cyclist, vehicle, etc.) are both relevant parameters, then, because the top speed of a pedestrian is certainly lower than the top speed of a vehicle, additional constraints can be included as part of the scenario specification to prevent concrete scenarios being generated where a pedestrian moves at 30 m/s. Note that these scenario-level constraints are provided in addition to the ODD compatibility matrices from Section II-A.

Finally, the ODD should guide the parameter selection process. In particular, the range of values assigned to each parameter should be based on the values permitted in the ODD. For example, if rainfall intensity is a parameter for the scenario, then the intensity values chosen for this parameter should cover the range of rainfall intensities listed in the ODD.

TABLE I  
PARAMETERS FOR PEDESTRIAN VARIANT EXAMPLE SCENARIO.

Property [unit]	# Values	Sample 1	Sample 2
Distance (ego to ped.) [m]	6	5	50
Ego Speed [m/s]	9	5	35
Ped. Speed [m/s]	6	2	2
Ped. Heading [deg]	8	135	135
Ped. Position w.r.t Ego [deg]	8	45	180
Ped. Sub-Type	6	Pedestrian	Cross. Guard
Ped. Height [m]	9	1.5	1.8
Ped. Silhouette [aspect ratio]	4	1:8-1:2	>1:2
Ped. Skin (Fitzpatrick Scale)	9	FP6 - Dark	FP1 - Pale
Ped. Hair Color	13	Brown	-
Ped. Hair Length [m]	7	0.30	0.00
Ped. Hair Style	15	Loose	Headscarf
Ped. Clothing Color	6	Patterned	Reflective
Ped. Clothing Style	11	Costume	Normal
Ped. Age [years]	6	0-5	20-60
Ped. Pose	6	Arms Up	Hunched
Ped. Carrying Object	16	-	Sign
Ped. Activity	25	Walking	Running
Ped. Angle of View	6	Left	Side Back

a) *Example - Parameterized Pedestrian Variant Scenario:* Continuing the illustrative example of a pedestrian variant, using the challenge hypothesis  $CH_0$ , we selected parameters that are related to the pedestrian’s appearance and position and motion of the object (relative to the ego vehicle). Since this logical scenario is focused strictly on the detectability of



pedestrian variants, for this example we chose not to include parameters related to environmental conditions or object occlusion. We have other logical scenarios that address these concerns directly and that consider some (modest) level of interaction between pedestrian appearance and environmental conditions; see Section III, Scenario 2. If further analysis and testing suggested that object variants and environmental conditions interacted strongly to challenge the system, further scenarios could be developed to investigate this interaction.

Table I shows the parameters selected for the pedestrian variant example scenario. Equivalence classes are used to define the values for each parameter, the number of equivalence classes for each parameter is reported in the table (exact values withheld for confidentiality). There were no scenario-level constraints. Columns “Sample 1” and “Sample 2” show example test cases produced with the CACTus method.

#### D. Step 4) Generate Concrete Scenarios

Given one or more scenario specifications, concrete scenarios are generated using a CT test case generation algorithm, such as the one implemented by ACTS [25]. To run the CT algorithm, each parameter from the scenario is translated into a test variable that can assume its specified values (e.g., equivalence classes). Then, relevant cells from the ODD compatibility matrix (see Step 1) and the scenario-level constraints (see Step 3) are translated into logical constraints that prevent the CT algorithm from generating outputs with invalid parameter combinations. The CT algorithm produces a set of concrete scenarios (test cases) that can be executed using the appropriate test infrastructure, such as a simulator.

##### a) Example - Concrete Pedestrian Variant Scenarios:

Using the ACTS tool with combinatorial strength  $t = 3$ , we generated 7133 concrete scenarios for the pedestrian variant logical scenario (see parameters in Table I). The number of generated concrete scenarios for other choices of  $t$  are given in Table II. The two sample columns in Table I contain values for two concrete scenarios generated by ACTS. Sample 1 is a child near the front corner of the ego vehicle with dark skin, brown loosely styled hair wearing a costume with their arms above their head standing such that their left side is visible to the ego vehicle. This first sample is quite plausible. However, in the spirit of challenging the system-under-test, some more ‘interesting’ samples were also generated. For instance, Sample 2 is an adult crossing guard with pale skin wearing a head covering running while hunched over, located 50 meters behind the ego vehicle.

#### E. Step 5) Assess Coverage

After generating the concrete scenarios, various coverage reports can be produced. First, since CT algorithms guarantee a minimum of  $t$ -way coverage, the combinatorial strength of the set of concrete scenarios can be directly reported. Second, coverage of the ODD can be measured by counting the occurrence of each element of the ODD in the combined set of concrete scenarios (across all logical scenarios). This coarse measure of ODD coverage provides some confidence that all

relevant aspects of the ODD will appear (at minimum of  $N$  times) in the resulting test suite. Other measures of coverage might include hazard or SOTIF trigger coverage (if such lists are available) or requirements coverage (if requirements exist to mitigate specific SOTIF triggers). Gaps in coverage are permissible provided they are reviewed and justified by experts familiar with the system and its ODD. If coverage is acceptable, then the set of generated concrete scenarios can be used to exercise the system-under-test. Otherwise, earlier steps should be re-visited.

### III. APPLICATION

We have applied CACTus to generate concrete simulation scenarios aimed at testing the object detection functions (see summary in Section II) of Motion’s AV, which will serve as a commercial ‘robotaxi’ to transport passengers in urban centers. This section summarizes the challenge scenarios we developed using CACTus, describes our use of NIST’s ACTS tool to generate concrete scenarios, and recounts some lessons from the industrial application of CACTus. At this time, we cannot report the results of executing the generated scenarios.

#### A. Summary of Logical Scenarios

To date, we have identified a total of eight challenge scenarios (some containing sub-scenarios). These scenarios and the size of their parameter spaces are reported in Table 2. The parameter space is described using the notation from [20] that describes both the number of parameters and the number of values for each parameter. For example,  $4^26^18^2$  corresponds to a scenario where two parameters have four values, one parameter has six values, and two parameters have eight values. The size of the parameter space is obtained by arithmetic (treating superscripts as exponentiation and adjacent numbers as multiplication); in this case  $4 \cdot 4 \cdot 6 \cdot 8 \cdot 8 = 6144$  possible combinations (disregarding logical constraints between parameters). The number of constraints provided to ACTS (to exclude invalid parameter combinations) is also reported.

Scenario 1 is focused on detecting variants of critical objects such as pedestrians, pedalcycles, and vehicles. The illustrative example in Section II gives the details of this scenario for pedestrians (Scenario 1a) while other sub-scenarios address pedalcycles (Scenario 1b) and vehicles (Scenario 1c). To challenge the perception system, we selected a wide range of object attributes, see Table 1.

Scenario 2 is focused on detecting objects in limited visibility conditions that impact the performance of one or more of the vehicle’s sensors. This includes the possibility of heavy (unexpected) precipitation, fog, dust, and reduced ambient lighting. Though the goal in this scenario is still to detect objects of interest, the emphasis is on the interaction between environmental conditions and attributes of the object, so we have focused the parameter choices on a sub-set of all possible object attributes (e.g., pedestrian with dark clothing in reduced lighting conditions). Scenario 3 is focused on the effect of reflective surfaces (e.g., a large glass window), which might contain reflected images of objects. For this scenario, the

relative position of the ego vehicle, other objects of interest, and the reflective surface are particularly important, so these were chosen as parameters.

Scenario 4 is intended to challenge the object tracking capability of the perception system by providing cases where objects split (e.g., a pedestrian gets out of a vehicle) or merge (e.g., a pedestrian picks up a child). There are several sub-scenarios considered that an AV is likely to encounter in an urban setting. Since the focus is on object tracking, the parameters for Scenario 4 address object motion and behavior and not appearance nor environmental conditions.

Scenario 5 considers partially occluded objects of interest, e.g., a pedestrian standing behind another object such as a mailbox such that only their shoulders and head are visible to the ego vehicle. The intent is to determine if the ML algorithm(s) performing object detection and classification are sensitive to specific parts of an occluded object; for example, a vehicle might only be correctly classified if its wheels are visible. The parameters for Scenario 5 focus on the geometric relationship between the ego vehicle, the object of interest, and the occluding object such as the extent of occlusion and the portion of the occluded object. Scenario 8 is related and considers the case where a previously unoccluded object becomes (temporarily) fully occluded. For example, if a pedestrian walks behind a parked car and then re-emerges.

Scenario 6 considers the case where the perception system must process a large number of objects near the ego vehicle. For example, many pedestrians cross the road in front of the vehicle. This condition might occur during rush hour in a busy urban center or if there many people suddenly exit a building (e.g., an emergency in a hotel). Parameters for this scenario consider the number and type of objects as well as the behavior of the group (random motion, uniform motion, etc.).

Scenario 7 is related to Scenarios 1 and 2 and considers the case where an object’s appearance is very similar to its surroundings such that it is difficult to detect. For example, a pedestrian wearing grey clothing standing near a concrete wall. Parameters of interest include the appearance and size of the object, the relative position of the object, and characteristics of the object’s surroundings.

### B. Concrete Scenario Generation

Using the eight logical scenarios above, we applied step 4 of the CACTus method to generate concrete test scenarios for various combinatorial strengths up to  $t = 6$ . Scenarios were generated using the ACTS tool (v3.1, using the IPOG algorithm [26] [27]) on an Asus ZenBook with a 4-core 11th Gen Intel i7 processor at 2.8 GHz and 16 GB of RAM. Table 2 reports the number of concrete scenarios generated and the execution time (to generate the scenarios using ACTS) in seconds. Let  $k$  be the total number of parameters for each scenario, then the maximum number of tests (if every parameter was to be combined with every parameter) is reported in the column  $t = k$ ; this allows us to compare the reduction in number of scenarios afforded by the use of CT for choices of  $t$ . In cases when ACTS did not complete execution, “DNF” is

reported. Additionally, scenarios 4b, 4c, and 5 used only five parameters each, so values for the case where  $t = 6$  cannot be generated (denoted “N/A :  $k = 5$ ”). Figure 3 plots the number of concrete scenarios generated for selected logical scenarios (other scenarios had similar plots); this figure shows that the number of concrete scenarios increases exponentially with the combinatorial strength (even with constraints).

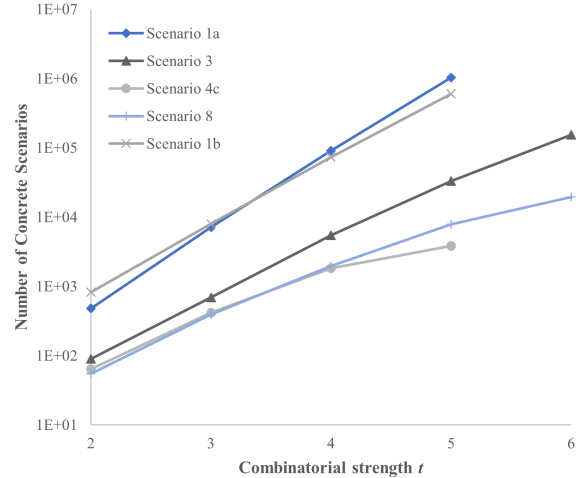


Fig. 3. Concrete scenarios for select logical scenarios.

### C. Lessons Learned

Next, we report several lessons learned from our experience applying CACTus to a commercial AV perception system.

a) *Use of System Failure Modes:* When selecting the logical scenarios and developing their challenge hypotheses, we found the system failure modes (as defined in an independent analysis) to be a particularly helpful area of engineering knowledge. More specifically, we examined the failure modes to find conditions that might be used to challenge the system-under-test. For example, Scenario 4 is a direct consequence of a hypothesized failure mode related to object tracking.

b) *Challenge Hypothesis and Scenario Scope:* While specifying the relevant parameters for each logical scenario, we repeatedly revisited the challenge hypothesis to ensure that the choice of parameters was aligned with our original intent of the scenario. We undertook scenario specification as a group activity and found the challenge hypothesis important for structuring our group discussions. Often one group member would propose a parameter and the other member would challenge the relevance of the parameter on the grounds that it was not aligned with the challenge hypothesis. While the purpose of the challenge hypothesis (per Section II) is indeed to structure these discussions, we would like to underscore its role in our thinking while parameterizing challenge scenarios.

c) *Hierarchical Taxonomy Improves Specification:* When specifying parameters, we valued having a structured taxonomy of the ODD that decomposed environment conditions and object types in a hierarchical manner. This meant that we could specify an entire category of conditions or objects very quickly

(rather than listing each variant on its own). For example, rather than listing all the variants of passenger cars (sedan, hatchback, coupe, etc.) we were able to specify the category ‘passenger car’ and use scripts to automatically populate the inputs to the ACTS tool for scenario generation. There is an opportunity to create tools (or extensions) that translate ODD specifications into inputs compatible with existing CT tools.

*d) Scenario Decomposition:* We initially attempted to specify Scenario 1 as a single scenario but ran into performance limitations with the ACTS tool while generating test cases. Our ODD is very detailed, and we believe the parameter space for Scenario 1 was simply too large for the tool to process within the time allotted. Instead, we were able to decompose the scenario into sub-scenarios by object type. While the runtime was still over three hours, we were able to generate concrete scenarios that covered our ODD. For users of CT tooling encountering performance problems, we recommend decomposing the parameter space along a boundary that is reasonable given the application area.

*e) Choice of Combinatorial Strength:* The value of  $t$  dictates the strength of the generated test suite. While empirical results for conventional software show that most defects are found when  $t = 6$  [20] we are not aware of any similar results for ML-based or autonomous systems. On one hand, if  $t$  is too small, then an important combinations might be missed. On the other hand, when  $t$  is too big, the size of the test suite is significant for an industrial scale ODD. During our project, we performed a literature review to determine common choices of  $t$  for autonomous systems and found that  $t = 2$  to  $t = 4$  has been used by some authors [18], [19], [28]–[31]; however, there do not appear to be any empirical studies supporting this choice. Such studies would greatly benefit practitioners who have to make firm decision about combinatorial strength.

#### IV. COMPARISON TO EXISTING METHODS

This section reviews related work on CT and scenario-based testing for autonomous systems and compares the CACTus method to existing methods and results.

Wotawa *et al.* describe a notation to define the input space of a complex system in the form of an ontology and demonstrate how such an ontology can be used for CT [19]. There attention is given to AVs, for instance defining both “car” and “truck” as types of “vehicle”, but the overall content of their paper is general and theoretical; the actual case study discussed focused on natural language sentence structure. This work was continued in [32]. In contrast, the CACTus method combines an ontology, expert knowledge, and CT to generate challenging scenarios. Moreover, in this paper we used CACTus to generate test cases for a commercial AV.

Tao *et al.* describe an ontology-based CT method to test an Autonomous Emergency Braking (AEB) system [29]. The scenarios were based on standard EuroNCAP scenarios [33], and the input space was almost entirely geometric in character, i.e., distances, speeds, and directions. This work was continued in [31]. In contrast, the CACTus method uses expert knowledge (expressed as a challenge hypothesis) to guide the development

of logical scenarios Furthermore, CACTus’ input space is not limited to geometric parameters and can also include discrete characteristics of agents in the scene, and the scene itself, such as weather conditions.

Tuncali *et al.* present a method to derive and execute test cases for AVs with ML-based components using a precise requirements-based framework [18]. Variations on these test cases were created using CT over parameters like pedestrian clothing color, vehicle color, and vehicle speeds. These test cases were executed automatically using their framework to find conditions that violate system requirements. Simulated Annealing was then used to generate adversarial test cases from the initial test results. In contrast, the CACTus method does not specify system requirements or iteratively generate adversarial test cases focusing instead on choosing challenging scenarios through grey-box knowledge of the system-under-test and choosing comprehensive parameter values.

Huang *et al.* present a case study applying pairwise ( $t = 2$ ) testing to a high-speed rail automatic train protection system (ATP) [30]. The test scenarios chosen focused on abnormal inputs to the ATP, with the parameters for each scenario being enumerations with on the order of 10 possible values per parameter. This approach successfully identified several faults in the ATP software. Huang *et al.*’s approach of choosing abnormal test inputs is similar to the CACTus method, but CACTus uses a much larger input space at a higher level of abstraction (environmental factors rather than system inputs), which is necessary to manage complexity of the operational environment of an AV vs. ATP.

Patel *et al.* present an approach using CT to evaluate ML models for bias and fairness [34]. Specifically, they examine an ML model making predictions about individuals financial or criminal status for bias related to protected characteristics such as age, race, religion, and gender. Although CACTus appears very dissimilar to this study, the ODD for our case study contained detailed characteristics for pedestrians overlapping with the protected characteristics identified in [34], including age, skin color and hair color, clothing, and hair length. Furthermore, although the ML components of the AV under study are not making predictions about an individual human, they are predicting whether a given region of space corresponds to a human based on sensor data. So, similarly to the justice system, bias for AVs is problematic.

The AVSC proposes a methodology to evaluate the behavioural competency of AVs in the context of their ODD using thresholds on defined metrics [35]. This approach focuses on the nominal top-level behavior of the vehicle under typical conditions, analogously to a human driver’s test. In contrast, CACTus aims to test vehicle system under unusual conditions.

In summary, like our CACTus method, several researchers have applied CT to structured descriptions of ODDs (ontologies etc.) for autonomous systems. However, the existing approaches are generally limited in two ways. First, they appear to combine all available parameters when generating test cases. When used at industrial scale, these methods will produce

TABLE II

CHALLENGE SCENARIOS, PARAMETER SPACE, NUMBER OF CONCRETE SCENARIOS GENERATED, AND ACTS RUNTIME; “DNF” INDICATES THE ACTS TOOL DID NOT FINISH EXECUTION WITHIN THE AVAILABLE TIME; THE  $t = k$  COLUMN WAS CALCULATED WITHOUT CONSIDERING CONSTRAINTS.

Scenario				Number of Concrete Scenarios (execution time in seconds)					
#	Name	Parameter Space	Number of Constraints	$t = 2$	$t = 3$	$t = 4$	$t = 5$	$t = 6$	$t = k$
1a	Pedestrian Variant	$4^1 6^7 7^1 8^2 9^3 11^1 13^1 15^1 16^1 25^1$	662	478 (4.2)	7,133 (39.2)	90,405 (604.5)	1,038,739 (13,266.6)	DNF	$3.1 \times 10^{17}$
1b	Vehicle Variant	$3^1 6^3 7^1 8^4 9^1 16^1 72^1$	2153	823 (8.5)	7,948 (34.3)	73,301 (285.1)	602,685 (3,251.5)	DNF	$1.9 \times 10^{10}$
1c	Pedalcycle Variant	$1^1 4^1 6^6 7^1 8^2 9^4 11^1 13^1 15^1 16^1$	1007	215 (3.8)	2,812 (15.9)	34,234 (178.1)	372,649 (2847.1)	DNF	$1.9 \times 10^{11}$
2	Limited Visibility	$3^1 4^3 5^2 6^3 7^1 8^2 9^2 28^1$	183	347 (2.9)	3,682 (16.1)	36,719 (163.5)	312,325 (2555.5)	DNF	$1.1 \times 10^{12}$
3	Reflective Surface	$2^1 3^3 4^3 6^1 8^2 9^2$	1	90 (0.8)	695 (2.6)	5,437 (18.2)	33,102 (85.3)	153,806 (406.5)	$1.1 \times 10^8$
4a	Object Split or Merge - Cyclist Falls of Bicycle	$1^1 6^4 8^2 9^1$	15	89 (1.3)	775 (2.7)	5,277 (12.5)	29,508 (52.1)	95,832 (182.0)	$8.3 \times 10^4$
4b	Object Split or Merge - Pedestrian Exits Vehicle	$4^1 5^1 6^1 8^2$	0	64 (0.6)	391 (1.1)	1,920 (3.0)	7,680 (9.1)	N/A: $k = 5$	$7.7 \times 10^3$
4c	Object Split or Merge - Pedestrian Picks up Object	$4^2 6^1 8^2$	1	64 (1.1)	416 (1.4)	1,831 (3.2)	3,840 (5.0)	N/A: $k = 5$	$6.1 \times 10^3$
4d	Object Split or Merge - Adult Picks Up Child	$4^3 6^1 8^2$	1	64 (1.0)	414 (1.6)	1,993 (4.1)	7,892 (10.5)	15,360 (21.5)	$2.5 \times 10^4$
5	Partially Occluded Object of Interest	$4^1 5^1 8^1 9^2$	0	81 (0.5)	651 (1.5)	3,240 (5.2)	12,960 (14.8)	N/A: $k = 5$	$1.3 \times 10^4$
6	Many Objects	$3^2 4^2 5^1 6^1 9^1$	2	55 (1.2)	285 (1.5)	1,253 (3.4)	5,083 (8.0)	16,060 (26.4)	$3.9 \times 10^4$
7	Low-Contrast Background	$2^2 4^2 5^1 6^1 7^2 9^1$	3	64 (1.1)	404 (2.1)	2,081 (6.0)	8,730 (17.2)	30,781 (64.2)	$8.5 \times 10^5$
8	Temporarily Occluded Object	$3^1 4^1 6^3 9^1$	2	55 (1.2)	393 (3.9)	1,971 (3.9)	7,828 (11.1)	30,781 (64.2)	$2.3 \times 10^4$
			<b>Total</b>	<b>2,489</b>	<b>25,998</b>	<b>255,982</b>	<b>2,334,021</b>	-	$3.3 \times 10^{17}$

extremely large test sets, even when CT is used to contain the growth. In contrast, CACTus combines expert judgement with grey-box knowledge of the system to intelligently select test parameters, resulting in a more focused set of test cases. Second, to our knowledge, none of the existing approaches have been applied at a commercial/industrial scale. Existing approaches use of CT to produce on the order of 100s of test cases whereas our results show many orders of magnitude more test cases for a commercial AV’s perception system.

## V. DISCUSSION

This paper has introduced the CACTus method for generating test suites that challenge autonomous systems. The use of a ‘challenge hypothesis’ is an important innovation that allows expert judgement to guide creation of test scenarios and reduce the parameter space. The method was applied to generate test cases for commercial AV’s perception system. Below we describe an alternative use of CACTus, the methods limitations, and then make suggestions for future work.

### A. Reversing CACTus to Measure Coverage

In this paper CACTus is presented as a ‘forward’ method to generate concrete scenarios, it could also be used in reverse to measure the functional coverage of an existing set of scenarios. In our experience, this alternative use case occurs frequently

in industry. The reverse method would, given a set of concrete scenarios and a specification for one or more corresponding logical scenarios, produce a measure of combinatorial strength for each logical scenario. Tools such as ACTS already have functions that assess combinatorial strength of a given set of test inputs. This would provide practitioners with a sense of whether a set of concrete scenarios derived via another process have adequate coverage of the ODD.

### B. Limitations

CACTus is intended to produce challenge scenarios for the system-under-test, i.e., to find unexpected behaviors or defects. However, like any engineering method, it has limitations. Specifically, it should not be used as the sole method to demonstrate that a system is adequate for its intended purpose. Other verification and validation methods, such as those recommended by ISO 26262 or ISO 21448 are still applicable. In particular, though CACTus provides a measure of ODD coverage and combinatorial strength at the concrete scenario level, it does not include any notion of coverage at the logical scenario level. Since logical scenarios are selected to challenge the system-under-test, it is likely that other scenarios (which might not appear challenging) are not included.



### C. Future Work

Next steps for the CACTus method include applying it to other systems to demonstrate its applicability and effectiveness more broadly, both in the area of AVs and other autonomous systems; determining how to integrate more sophisticated CT methods, such as mixed strength CT to further reduce the size of the generated test suite [36]; conducting empirical studies to determine a suitable combinatorial strength for autonomous system testing; developing tools to support scenario specification activities that take account of large or complex operational domains, such as deriving challenge scenarios from defeater nodes in an EA argument [37]; specifying the ODD as a formal ontology; executing the generated tests to determine effectiveness in finding faults when compared to other test-generation methods, such as random testing or adaptive random testing [38]; and using insight gained from test results to improve the underlying system and as evidence in an argument for the safety of the system.

### REFERENCES

- [1] ISO/TC 22/SC 32, "ISO/PAS 21448:2019 - Road vehicles - Safety of the intended functionality," International Organisation for Standardisation, 2019.
- [2] "ANSI/UL 4600 - Standard for Safety for the Evaluation of Autonomous Vehicles and Other Products," 2020.
- [3] R. Ashmore, R. Calinescu and C. Paterson, "Assuring the Machine Learning Lifecycle: Desiderata, Methods, and Challenges," *ACM Computing Surveys*, vol. 54, no. 5, pp. 111:1-111:39, 2021.
- [4] P. Kohli and A. Chadha, "Enabling Pedestrian Safety Using Computer Vision Techniques: A Case Study of the 2018 Uber Inc. Self-driving Car Crash," in *Advances in Information and Communication. FICC 2019*, 2019.
- [5] P. Penmetsa, P. Sceinidashtegol, A. Musaev, E. Kofi Adanu and M. Huddnall, "Effects of the autonomous vehicle crashes on public perception of the technology," *IATSS Research*, vol. 45, no. 4, pp. 485-492, 2021.
- [6] "ISO 26262 - Road Vehicles - Functional Safety," International Organisation for Standardization, 2018.
- [7] P. Koopman, "How Safe is Safe Enough?: Measuring and Predicting Autonomous Vehicle Safety," Independently Published, 2022.
- [8] P. Koopman, "The Heavy Tailed Safety Ceiling," in *Automated and Connected Vehicle Systems Testing Symposium*, 2018.
- [9] P. Koopman, "Challenges in Autonomous Vehicle Testing and Validation," *SAE International Journal of Transportation Safety*, 2016.
- [10] Y. Tian, K. Pei, S. Jana and R. Baishakhi, "DeepTest: automated testing of deep-neural-network-driven autonomous cars," in *International Conference on Software Engineering*, New York, NY, USA, 2018.
- [11] L. Ma, F. Juefel-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu, J. Zhao and Y. Wang, "DeepGauge: multi-granularity testing criteria for deep learning systems," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE 2018)*, 2018.
- [12] Y. Sun, X. Huang, D. Kroening, J. Sharp, M. Hill and R. Ashmore, "Structural Test Coverage Criteria for Deep Neural Networks," *ACM Transactions on Embedded Computing Systems*, vol. 18, no. 5s, pp. 1-23, 2019.
- [13] Y. Dong, P. Zhang, J. Wang, S. Liu, J. Sun, J. Hao, X. Wang, J. S. Dong and D. Ting, "There is Limited Correlation between Coverage and Robustness for Deep Neural Networks," *arXiv preprint arXiv:1911.05904*, 2019.
- [14] R. Ashmore and A. Banks, "The Utility of Neural Network Test Coverage Measures," in *The AAAI's Workshop on Artificial Intelligence Safety*, 2021.
- [15] Z. Li, X. Ma, X. C and C. Cao, "Structural Coverage Criteria for Neural Networks Could Be Misleading," in *International Conference on Software Engineering: New Ideas and Emerging Results*, 2019.
- [16] L. Ma, F. Xue, B. Li, L. Li, Y. Liu and J. Zhao, "DeepCT: Tomographic Combinatorial Testing for Deep Learning Systems," in *International Conference on Software Analysis, Evolution and Reengineering*, 2019.
- [17] L. Ma, F. Zhang, M. Xue, B. Li, Y. Liu, J. Zhao and Y. Wang, "Combinatorial Testing for Deep Learning Systems," in *arXiv preprint*, 2019.
- [18] Tuncali, Fainekos, Prokhorov, Ito and Kapinski, "Requirements-driven Test Generation for Autonomous Vehicles with Machine Learning Components," *IEEE Transactions on Intelligent Vehicles*, 2019.
- [19] F. Wotawa and Y. Li, "From Ontologies to Input Models for Combinatorial Testing," *IFIP International Conference on Testing Software and Systems*, vol. 11146, pp. 155-170, 2018.
- [20] D. Kuhn, R. Kacker, Y. Lei and D. Simos, "Input Space Coverage Matters," *Computer (IEEE Computer)*, vol. 53, no. 1, pp. 37-44, 2020.
- [21] K. Czarniecki, "Operational World Model Ontology for Automated Driving Systems - Part 2: Road users, animals, other obstacles and environmental conditions," *Waterloo Intelligent Systems Engineering Lab (WISE) Report*, University of Waterloo, 2018.
- [22] "AVSC Best Practice for Describing an Operational Design Domain: Conceptual Framework and Lexicon," *Automated Vehicle Safety Consortium*, 2020.
- [23] "PAS 1883:2020 - Operational Design Domain (ODD) taxonomy for an automated driving system (ADS) - Specification," *British Standards Institution*, 2020.
- [24] K. Czarniecki, "Operational World Model Ontology for Automated Driving Systems - Part 1: Road Structure," *Waterloo Intelligent Systems Engineering Lab (WISE) Report*, University of Waterloo, 2018.
- [25] R. N. Kacker, "ACTS: A Combinatorial Test Generation Tool," in *Proceedings of Sixth IEEE International Conference on Software Testing, Verification and Validation ICST 2013*, Luxembourg, 2013.
- [26] Y. Lei and K. Tai, "In-parameter-order: a test generation strategy for pairwise testing," in *Proceedings Third IEEE International High-Assurance Systems Engineering Symposium*, 1998.
- [27] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun and J. Lawrence, "IPOG: A General Strategy for T-Way Software Testing," in *14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)*, Tucson, AZ, USA, 2007.
- [28] D. R. Kuhn, R. N. Kacker and Y. Lei, "Introduction to Combinatorial Testing," *Chapman and Hall/CRC*, 2013.
- [29] Tao, Li, Wotawa, Felbinger and Nica, "On the Industrial Application of Combinatorial Testing for Autonomous Driving Functions," *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2019.
- [30] R. Huang, C. Rao, Y. Lei, J. Guo and Y. Zhang, "Applying Combinatorial Testing to High-Speed Railway Automatic Train Protection System," *IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2022.
- [31] Y. Li, J. Tao and F. Wotawa, "Ontology-based test generation for automated and autonomous driving functions," *Information and Software Technology*, vol. 117, 2020.
- [32] F. Kluck, Y. Li, M. Nica, T. Jianbo and F. Wotawa, "Using Ontologies for Test Suites Generation for Automated and Autonomous Driving Functions," *IEEE International Symposium on Software Reliability Engineering Workshops*, 2018.
- [33] Euro NCAP, "TEST PROTOCOL - AEB systems," *EUROPEAN NEW CAR ASSESSMENT PROGRAMME*, Brussels, Belgium, 2017.
- [34] A. Patel, J. Chandrasekaran, Y. Lei, R. Kacker and D. R. Kuhn, "A Combinatorial Approach to Fairness Testing of Machine Learning Models," *IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2022.
- [35] SAE AVSC, "AVSC00008202111 - AVSC Best Practice for Evaluation of Behavioral Competencies for Automated Driving System Dedicated Vehicles (ADS-DVs)," 2021.
- [36] M. Ozcan, "Applications of Practical Combinatorial Testing Methods at Siemens Industry Inc., Building Technologies Division," in *IEEE International Conference on Software Testing, Verification and Validation Workshops*, 2017.
- [37] J. B. Goodenough, C. B. Weinstock and A. Z. Klein, "Eliminative Argumentation: A Basis for Arguing Confidence in System Properties," *Carnegie Mellon University - Software Engineering Institute*, Pittsburgh, United States, 2015.
- [38] T. Y. Chen, H. Leung and I. K. Mak, "Adaptive Random Testing," *Advances in Computer Science - ASIAN 2004. Higher-Level Decision Making*, pp. 320-329, 2005.