

# Utilizing Ontologies for Combinatorial Testing

Franz Wotawa

*Institute of Software Engineering and Artificial Intelligence*

*Graz University of Technology*

Graz, Austria

wotawa@tugraz.at

**Abstract**—Test case generation using combinatorial testing requires an input model comprising parameters and their ranges of possible values, i.e., their domains, which might be appropriate for ordinary applications but not for more sophisticated application domains like autonomous systems. Therefore, we discuss the use of ontologies for modeling and its utilization for combinatorial testing. We focus on ontologies describing the interface between the system under test and its environment, discuss the rationale behind such an approach, and introduce the basic algorithm for mapping ontologies to input models. We further consider modeling challenges and provide some initial modeling principles.

**Index Terms**—Ontology-based testing, combinatorial testing, ontology modeling.

## I. INTRODUCTION

Quality assurance, i.e., showing that a system fulfills given quality criteria, is an important task of system and software engineering. Testing, i.e., executing the system or software considering specific input stimuli and checking its behavior for deviations from expectations, is part of quality assurance focusing on the detection of failures caused by faults. For saving costs and other resources, test automation has been an important research topic for several decades, leading to test case generation techniques like combinatorial testing (CT) [1]–[3]. In CT, the idea is to find critical interactions of values for inputs that reveal an unexpected behavior. Instead of considering all possible input combinations, which is unfeasible to be carried out for real systems and software, CT uses a fixed number  $k$  where for any parameter set of size  $k$  all combinations are considered. This leads to substantially fewer test cases and makes CT applicable for practice. For a survey on CT and its tools and methods, we refer to [4] and [5], respectively. Several papers already showed success stories of CT when being applied to relevant systems, e.g., see [6] and [7]. Notably, [6] showed that a smaller number of parameter combinations of less or equal to 6 is good enough to reveal all faults in different software and systems. There are several tools available, including ACTS [8] and CAGen [9], and besides ordinary software and system testing, CT has been used for, e.g., security testing [10].

This paper is part of the A-IQ Ready project that has received funding within the CHIPS JU in collaboration with the European Union's Horizon Framework Programme and National Authorities under grant agreement No. 101096658. The work was partially funded by the Austrian Federal Ministry of Climate Action, Environment, Energy, Mobility, Innovation and Technology under the FFG project FO999896574.

The input for test generation using CT is a set of parameters, a domain for any parameter, a set of constraints a test case must fulfill, and the strength  $k$ , i.e., the size of any subset of parameters where all combinations must be considered. This input is called an input model. The output is a set of tests, i.e., an assignment of values from the corresponding domain to any parameter fulfilling all criteria of CT, i.e., fulfilling the constraints and the presence of all combinations of values for any subset of parameters of the given size. In many situations, coming up with an input model for CT is trivial, e.g., when all inputs are of basic type and unstructured, but in other cases, coding the right information to fit an input model is not. For example, when a system has only one input, e.g., a text, but we want to find interactions of text fragments for detecting faults. Such a situation occurs, e.g., in compiler or security testing. Another example domain is autonomous driving, where the vehicle under test interacts with other cars and pedestrians over time. Hence, there is a need for other approaches for formulating knowledge that can be used as input for combinatorial testing. Indeed, this challenge is not new and has already been tackled. Earlier work of Satish and colleagues [11]–[13] used different UML diagrams for modeling from which the input model, can be extracted. With a similar idea, Wotawa and Li [14] introduced an approach where ontologies for conceptualizing knowledge are used for CT. This approach (particularly the conversion algorithm) was improved and applied to testing autonomous driving and automated driver assistance systems [15]. Most recently, the input model obtained from ontologies was used to compare different testing methods [16]. Furthermore, there are publications showing that the approach can be used in other domains, like security testing [17] and compiler testing [18]. It is worth noting that there are also papers dealing with different modeling paradigms, including hierarchical modeling [19], [20] and tree-structured models [21].

This paper also relies on ontologies as a knowledge representation paradigm for combinatorial testing. However, in contrast to previous work, we focus on the modeling part and discuss the rationale behind utilizing ontologies and how they should be constructed. We give some illustrative examples and present open issues and challenges. Furthermore, we introduce a slightly modified algorithm for converting ontologies into input models. Hence, the contributions of this paper are (i) improving existing algorithms for ontology conversion and (ii) discussing modeling issues.

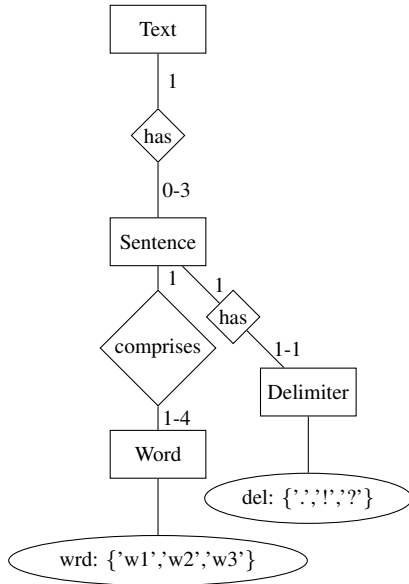


Fig. 1. A text ontology

We organize the paper as follows. In Section II we start presenting the foundations of ontologies and the conversion process. Afterward, in Section III we discuss modeling and how an ontology for testing should be constructed. Finally, we conclude the paper in Section IV.

## II. ONTOLOGY FOUNDATIONS

Feilmayr et al. [22] defines ontologies as formal, explicit specifications of shared conceptualizations that are characterized by high semantic expressiveness required for increased complexity, which captures concepts and their relationships. From this informal definition, we can extract the terminologies of concepts and their relationships, which – at least – seems to be close to similar formal models like entity-relationship diagrams from the domain of databases or UML class diagrams. In the following, we make use of these similarities. However, let us first formalize the definition of ontologies taken from [14] under the restriction assumption of only considering composition and inheritance as the only types of relationships.

An *ontology* is a tuple  $(C, A, D, \omega, R, \tau, \psi)$  where  $C$  is a finite set of concepts,  $A$  is a finite set of attributes which characterize concepts,  $D$  is a finite set of domain elements. Function  $\omega$  maps concepts to a set of tuples specifying the attribute and its domain elements.  $R$  is a finite set of tuples stating that two concepts are related. Function  $\tau$  assigns a type (i.e., composition ( $c$ ) or inheritance ( $i$ )) to each relation. Function  $\psi$  maps composition relations to their minimum and maximum arity, specifying how many composer concepts a composer concept may comprise, ranging from 0 or 1 to any natural number, which is known when developing the ontology.

Let us consider developing a *text* ontology as an example. A text comprises sentences. Sentences, themselves,

have 1 or more words that are terminated by a delimiter like a '.' or '!'. Furthermore, we can represent words as a sequences of characters etc. However, let us only focus on the concepts *text*, *sentence*, *word*, and *delimiter*. In Figure 1 we depict a graphical representation of such a text ontology. In this ontology we cover the basic concepts and their relationships. Note that the relationships in the figure already have their arity attached. We assume that a text has 0 to 3 sentences, a sentence comprises 1–4 words and exactly 1 delimiter. For the concepts *word* and *delimiter*, we also added attributes with their domains.

Formally, we can express the text ontology  $(C_T, A_T, D_T, \omega_T, R_T, \tau_T, \psi_T)$  as follows:

$$C_T = \{\text{Text}, \text{Sentence}, \text{Word}, \text{Delimiter}\}$$

$$A_T = \{\text{wrds}, \text{del}\}$$

$$D_T = \{w1, w2, w3, '.', '!', '?'\}$$

$$\omega_T(\text{Text}) = \{\}$$

$$\omega_T(\text{Sentence}) = \{\}$$

$$\omega_T(\text{Word}) = \{(\text{wrds}, \{w1, w2, w3\})\}$$

$$\omega_T(\text{Delimiter}) = \{(\text{del}, \{', '!', '?'\})\}$$

$$R_T = \left\{ \begin{array}{l} (\text{Text}, \text{Sentence}), \\ (\text{Sentence}, \text{Word}), \\ (\text{Sentence}, \text{Delimiter}) \end{array} \right\}$$

$$\tau_T(\text{Text}, \text{Sentence}) = c$$

$$\tau_T(\text{Sentence}, \text{Word}) = c$$

$$\tau_T(\text{Sentence}, \text{Delimiter}) = c$$

$$\psi_T(\text{Text}, \text{Sentence}) = (0, 3)$$

$$\psi_T(\text{Sentence}, \text{Word}) = (1, 4)$$

$$\psi_T(\text{Sentence}, \text{Delimiter}) = (1, 1)$$

Note that any relationship  $(c, c')$  is considered to be directed, i.e., the concept  $c$  is composed of a number of concepts  $c'$ . Hence, we assume that for any instance of concept  $c$  there are instances of concept  $c'$  unless the minimum arity is set to 0, where we might have no such instance. Because of defining relationships to be directed, we can define a *root concept* as a concept that is never on the right side of a relationship and *leaf concepts* to be concepts that are on the right side and never on the left side of a relationship. We further restrict ontologies to be *tree-structured*, i.e., do not comprise any cycles in the structure and to be connected, i.e., every concept needs to be element of at least one relationship. In this case, we do have only one root concept and some leaf concepts. We call such an ontology *well-formed*.

To use ontologies as inputs for CT, we need to convert them into a CT *input model*  $(P, D, C)$  comprising a set of parameters  $P$ , a domain for each parameter from  $D$ , and a set of constraints  $C$  a CT test suite must satisfy. This conversion should respect the underlying semantics behind the concepts and their relationships. For attributes of concepts, this conversion is – more or less – straightforward. Because attributes present inputs, they can be mapped to parameters. In order to make each parameter unique, we also add the concept name to them. Hence, for our *text* ontology, the attributes of concepts *word* and *delimiter* are mapped to *word.wrds* and *delimiter.del* respectively. Clearly, the domains of the parameters originate from the domains of their

corresponding attributes. However, we also add a new element  $\epsilon$  representing the empty value to each domain.

---

**Algorithm 1** `onto2im_h` ( $c, O$ )

---

**Require:** A concept  $c$  of a well-formed ontology  $O$ .

**Ensure:** An input model  $(V, \{D_{v_1}, \dots\}, C)$  for  $c$ .

```

1: Let  $V$  and  $C$  be empty sets.
2: for all attributes  $a \in \omega(c)$  do
3:   Add  $c.a$  to  $V$ .
4:   Let  $D_{c.a}$  be  $\text{dom}(c, a)$ .
5: end for
6: if  $c$  is not a leaf concept then
7:   for all relationships  $(c, c') \in R$  with type  $\tau(c, c') = c$  do
8:     Call onto2im( $c', O$ ) and store the input model in  $(V', \{D'_{v_1}, \dots\}, C')$ .
9:     Add  $\epsilon$  to all the variable domains in  $\{D'_{v_1}, \dots\}$ .
10:    Assume that  $\psi(c, c') = (n, m)$ 
11:    for  $i = 1$  to  $m$  do
12:      Add the content of  $V'$ ,  $\{D'_{v_1}, \dots\}$ , and  $C'$  to  $V$ ,  $\{D_{v_1}, \dots\}$ , and  $C$  respectively, but rename all elements  $x$  of  $V'$  to  $c.i.x$  before!
13:    end for
14:    for all attributes  $a$  in  $V'$  do
15:      if  $n > 0$  then
16:        Add  $\bigvee_{\{r_1, \dots, r_n\} \subseteq \{c.1.a, \dots, c.m.a\}} (r_1 \neq \epsilon \wedge \dots \wedge r_n \neq \epsilon)$  to  $C$ .
17:      end if
18:    end for
19:  end for
20: end if
21: return  $(V, \{D_{v_1}, \dots\}, C)$ 

```

---

The rationale behind adding  $\epsilon$  originates from the handling of the relationships. For this paper, we focus only on composition. Let us illustrate the handling using our `text` ontology. We know that a sentence comprises words. As indicated, each sentence has 1 to 4 words. Hence, a sentence comprises either 1, 2, 3 or 4 words. To represent this, we introduce a new parameter for each of these words using a similar naming schema than before. Hence, we introduce parameters `sentence.1.word.wrd`, `sentence.2.word.wrd`, `sentence.3.word.wrd`, `sentence.4.word.wrd`, which covers all potential words in a sentence. However, not all of them might be present. Hence, by assigning the value  $\epsilon$ , a word has no value in a test case. In addition of  $\epsilon$ , we also need to add a constraint. In the particular case, we know that there must be at least one word that has not  $\epsilon$  assigned in a test case, which needs to be covered in a constraint.

Note also that the proposed algorithm always expands the name of attributes when going upwards the tree until reaching the root concept. For example, one parameter obtained from the `text` ontology is `text.1.sentence.2.word.wrd`. Each parameter name represents the path from the root to the declaration of an attribute.

The `onto2im` algorithm takes a well-formed ontology  $O$  and computes the input model for combinatorial testing. This computation can be done starting from the root concept  $r$ . Hence, `onto2im` calls a helper function `onto2im_h` on  $r$  and  $O$  that is implemented in Algorithm 1 and returns its result. Note that we do not handle inheritance in this version of the algorithm due to space limitations.

### III. MODELING

Besides providing a fast algorithm that compiles an ontology into a CT input model having a reasonable number of parameters, each with smaller domains, we need an ontology, which requires capturing those parts of the environment of the system or software under test that maps to an input. Modeling is never easy nor providing a one-to-one representation of the world. Hence, we need a modeling methodology that supports testers in ontology development.

#### a) Identify the relevant concepts and their relationships:

Relevant concepts are concepts that either are bringing together other concepts, like `sentence` and `word` in our `text` ontology, or represent something that holds attributes, which map to inputs or parts of inputs, e.g., `word`. Try to come up with an ontology using the concepts and relationship that is well-formed. This step is close to ordinary modeling already well-known to be carried out in other domains, like data modeling for relational databases or coming up with class diagrams for object-oriented software. Note that a simple ontology might only comprise one concept where all input parameters of a system or software are represented by their corresponding attributes. Such a simple representation might be sufficient for testing. Unfortunately, sufficiency depends not on the number of inputs. In the case of a compiler where we only use one file or textual representation of a program as input, we still might want to use ontologies for representing the grammar of the input program. In this way, we are able to generate different programs as inputs easily (see [18]).

#### b) Introduce testing specific root concepts:

If necessary, come up with a root concept that represents a scenario or something similar. For example, in the automotive domain, where we want to test an autonomous vehicle, we may state a concept `scenario` that aggregates two sub-concepts `static_part` and `dynamic_part`, where the `static_part` comprises all elements that are used to compose a static structure in a scenario like streets, traffic signs, etc. The `dynamic_part` covers other vehicles and pedestrians with their attributes like initial location and speed.

#### c) Handling time using ontologies:

Even in the case of simple systems, we might want to use ontologies. One example is when there is a need to provide test stimuli over a discrete number of time steps. Let us assume we have a simple ontology comprising a system  $S$  with two attributes  $a$  and  $b$ . If we need to provide inputs at different time steps, we can introduce a new root concept, say  $T$ , and state that we need 1 to 5 instances of  $S$ . For such an ontology, have a look at Table I. Note that in this input model, the constraint states

TABLE I  
THE INPUT MODEL OF THE SIMPLE ONTOLOGY FROM FIG. 2.

Parameter	Domain
T.1.S.a	{0,1}
T.1.S.b	{0,1}
...	...
T.5.S.a	{0,1}
T.5.S.b	{0,1}
Constraint:	
$T.1.S.a \neq \epsilon \wedge T.1.S.b \neq \epsilon \wedge \dots \wedge T.5.S.b \neq \epsilon$	

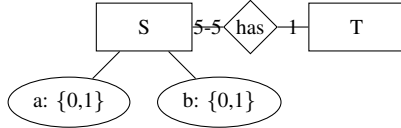


Fig. 2. A simple ontology where T is the root concept

that  $\epsilon$  shall not be used in this case to eliminate an instance in any test.

The three proposed modeling principles should be seen as inputs for a general modeling methodology, which needs a more sophisticated discussion with illustrative examples that would go beyond the purpose of this paper. However, from the three modeling principles, we see that using ontologies increases flexibility. For example, it is easy to increase the number of time steps to be considered for scenarios. We only need to change 5-5 to 10-10 for our simple ontology example. Moreover, the more general an ontology is, the more likely its content can be reused. Hence, there is a potential of reducing the overall resources required for test case generation. Nevertheless there are also open issues and challenges worth being raised. First of all, there are limiting assumptions introduced for ontologies (e.g., the different types of relationships, well-formed ontologies, etc.). Second, there is currently only limited tool support for conversion, and the way ontologies are described is not standardized. Finally, we may obtain input models that generate a huge number of test cases even for combinatorial strengths of 2 or 3. In this case, test execution may not be feasible anymore.

#### IV. CONCLUSIONS

In this short paper, we discuss the use of ontologies for combinatorial testing. In particular, we define an ontology, show how it can be automatically mapped into a combinatorial testing input model, and raise the need for a general modeling methodology. We further discuss some principles behind modeling, showing that in some cases, ontology positively impacts re-use and enables the easy handling of time sequence models. Finally, we introduce open questions and issues to be tackled in future research activities.

#### REFERENCES

- [1] R. Kuhn, Y. Lei, and R. Kacker, "Practical combinatorial testing: Beyond pairwise," *IT Professional*, vol. 10, no. 3, pp. 19–23, 2008.
- [2] D. Kuhn, R. Kacker, and Y. Lei, *Introduction to Combinatorial Testing*, ser. Chapman & Hall/CRC Innovations in Software Engineering and Software Development Series. Taylor & Francis, 2013.

- [3] D. R. Kuhn, R. Bryce, F. Duan, L. S. Ghandehari, Y. Lei, and R. N. Kacker, "Combinatorial testing: Theory and practice," in *Advances in Computers*, 2015, vol. 99, pp. 1–66.
- [4] C. Nie and H. Leung, "A survey of combinatorial testing," *ACM Comput. Surv.*, vol. 43, no. 2, pp. 11:1–11:29, Feb. 2011.
- [5] S. K. Khalsa and Y. Labiche, "An orchestrated survey of available algorithms and tools for combinatorial testing," in *25th International Symposium on Software Reliability Engineering*, 2015, pp. 323–334.
- [6] D. Kuhn, R. Kacker, Y. Lei, and J. Hunter, "Combinatorial software testing," *Computer*, pp. 94–96, August 2009.
- [7] J. Petke, S. Yoo, M. B. Cohen, and M. Harman, "Efficiency and early fault detection with lower and higher strength combinatorial interaction testing," in *Proc. of the 9th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'13)*, 2013, pp. 26–36.
- [8] L. Yu, Y. Lei, R. Kacker, and D. Kuhn, "Acts: A combinatorial test generation tool," in *Software Testing, Verification and Validation (ICST)*, 2013 *IEEE Sixth International Conference on*, 2013, pp. 370–375.
- [9] M. Wagner, K. Kleine, D. E. Simos, R. Kuhn, and R. Kacker, "Cagen: A fast combinatorial test generation tool with support for constraints and higher-index arrays," in *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2020, pp. 191–200.
- [10] J. Bozic, B. Garn, I. Kapsalis, D. Simos, S. Winkler, and F. Wotawa, "Attack pattern-based combinatorial testing with constraints for web security testing," in *IEEE International Conference on Software Quality, Reliability & Security (QRS)*, Vancouver, Canada, August 3–5 2015.
- [11] P. Satish, M. Basavaraja, M. S. Narayan, and K. Rangarajan, "Building combinatorial test input model from use case artefacts," in *10th IEEE International Conference on Software Testing, Verification and Validation Workshops*, 2017, pp. 220–228.
- [12] P. Satish, K. Sheeba, and K. Rangarajan, "Deriving combinatorial test design model from uml activity diagram," in *6th IEEE International Conference on Software Testing, Verification and Validation Workshops*, 2013, pp. 331–337.
- [13] P. Satish, A. Paul, and K. Rangarajan, "Extracting the combinatorial test parameters and values from uml sequence diagrams," in *7th IEEE International Conference on Software Testing, Verification, and Validation Workshops*, 2014, pp. 88–97.
- [14] F. Wotawa and Y. Li, "From ontologies to input models for combinatorial testing," in *Testing Software and Systems - 30th IFIP WG 6.1 International Conference, ICTSS 2018, Cádiz, Spain, October 1–3, 2018, Proceedings*, ser. LNCS, vol. 11146. Springer, 2018, pp. 155–170. [Online]. Available: [https://doi.org/10.1007/978-3-319-99927-2\\_14](https://doi.org/10.1007/978-3-319-99927-2_14)
- [15] Y. Li, J. Tao, and F. Wotawa, "Ontology-based test generation for automated and autonomous driving functions," *Information and Software Technology*, vol. 117, 2020.
- [16] F. Klück, Y. Li, J. Tao, and F. Wotawa, "An empirical comparison of combinatorial testing and search-based testing in the context of automated and autonomous driving systems," *Inf. Softw. Technol.*, vol. 160, 2023. [Online]. Available: <https://doi.org/10.1016/j.infsof.2023.107225>
- [17] J. Bozic, Y. Li, and F. Wotawa, "Ontology-driven security testing of web applications," in *IEEE International Conference On Artificial Intelligence Testing, AITest 2020, Oxford, UK, August 3–6, 2020*. IEEE, 2020, pp. 115–122. [Online]. Available: <https://doi.org/10.1109/AITest49225.2020.00024>
- [18] Y. Li and F. Wotawa, "On using ontologies for testing compilers," in *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2020, pp. 181–184.
- [19] L. Kampel, B. Garn, and D. E. Simos, "Combinatorial methods for modelling composed software systems," in *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2017, pp. 229–238.
- [20] J. Czerwonka, "Pairwise testing in the real world: Practical extensions to test-case scenarios," in *Proceedings of 24th Pacific Northwest Software Quality Conference, Citeseer*. Citeseer, 2006, pp. 419–430.
- [21] T. Kitamura, A. Yamada, G. Hatayama, C. Artho, E.-H. Choi, N. T. B. Do, Y. Oiwa, and S. Sakuragi, "Combinatorial testing for tree-structured test models with constraints," in *2015 IEEE International Conference on Software Quality, Reliability and Security*, 2015, pp. 141–150.
- [22] C. Feilmayr and W. Wöb, "An analysis of ontologies and their success factors for application to business," *Data & Knowledge Engineering*, pp. 1–23, 2016.