Scenario Sampling for Cyber Physical Systems using Combinatorial Testing

Akihisa Yamada*, Clovis Eberhart*, Fuyuki Ishikawa*, and Nian-Ze Lee†
* National Institute of Informatics, Tokyo, Japan
akihisayamada@nii.ac.jp, clovis.eberhart@gmail.com, f-ishikawa@nii.ac.jp
† National Taiwan University, Taipei, Taiwan
nianzelee@gmail.com

***Abstract*—Physical and continuous aspects are inevitable in cyber physical systems like automated driving systems. Despite the success of combinatorial testing on discrete systems, there is a fundamental challenge in applying combinatorial testing techniques when continuous parameters are involved. This extended abstract presents an initial step towards applying combinatorial testing to systems in which discrete and continuous parameters are mixed. We define a generic XML-based language for describing the test space of such systems and provide a prototype implementation to generate test cases, using externally the combinatorial test tool PICT.**

***Index Terms*—Cyber physical systems, Combinatorial testing, Random testing**

I. Motivation

When testing *cyber physical systems* such as automated driving systems, it is inevitable to choose real values for physical quantities. Consider an oversimplified automated driving system, where `ego` vehicle is given an initial position `x` and velocity `v` (which are real numbers) and there are several obstacles in some positions, possibly static or moving forward or backward at some velocity. An exhaustive testing of such a physical system is not just infeasible but *impossible*: one will need *uncountably* many test cases (scenarios) to cover the possibility of any real-valued parameter.

Random testing [4] provides an obvious way of choosing (i.e., sampling) real values from bounded real intervals and, under a well-chosen distribution, from unbounded continuous spaces. For the purpose of the quality assurance of safety-critical systems, however, the "random" nature of random testing might be undesirable; in particular, there is no agreed notion of coverage that the random testing literature provides.

Combinatorial testing [5], on the other hand, provides a well-defined quantitative coverage criterion—the combinatorial coverage—when the systems under test have only discrete input parameters. We expect combinatorial coverage to be still a useful criterion for cyber physical systems when continuous parameters are appropriately *partitioned*; for instance, Fig. 1 shows four out of nine test cases covering all pairwise combinations of obstacles which can be in front or in back, static or moving forward or backward.

The authors are supported by ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603), JST. The work is done during N.L.'s internship at National Institute of Informatics, Tokyo, Japan.

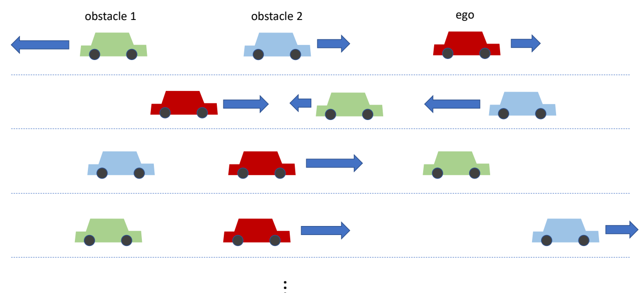

Fig. 1. First four test cases of a pairwise test suite generated using the combinatorial testing tool PICT.

In this work, we propose a general language for describing test scenarios with discrete and continuous parameters mixed, and provide a prototype implementation that generates concrete test cases from such a scenario description, by combining both the combinatorial testing approach and random testing approach. We have implemented a prototype scenario sampler, available at https://github.com/ERATOMMSD/scenario_sampler_code. This implementation externally calls the combinatorial testing tool PICT [3] for generating pairwise covering test suites. In collaboration with our industry partner, we actually used the scenario sampler for testing a prototype implementation of a simulator of an automotive system.

II. Syntax and Semantics

Here we define the language of the Scenario Sampling Script. The language constitutes of the following three layers:

$$\begin{aligned}
\ell 1 &::= \mathit{text} \mid \texttt{<text>}\mathit{text}\texttt{<text>} \\
&\quad\mid \texttt{<random min="}\mathit{real}\texttt{" max="}\mathit{real}\texttt{"/>} \\
\ell 2 &::= \ell 1 \mid \texttt{<choice>}\ell 1^{+}\texttt{</choice>} \\
\ell 3 &::= \ell 2 \mid \texttt{<repeat>}\ell 3^{+}\texttt{</repeat>}
\end{aligned}$$

By a layer-k script we mean a sequence of elements from the corresponding layer. In the following, we describe the role of each layer.

Layer 1: The first layer allows `<random>` tags, which represent real values taken from the range specified by attributes `min` and `max`, as well as plain text, which may be enclosed in `<text>...</text>` tags in order to clarify the structure. The structure of text is voluntarily kept ambiguous

```
"scenario": {
  "ego": { "x": -0.110019, "v": 83.180578 },
  "obstacles": [
    { "x": -30.431152, "v": 26.277911 },
    { "x": 45.284353, "v": 0 }
  ]
}
```

Fig. 2. A concrete “scenario”, where the positions and velocities of “ego” and two “obstacles” are unambiguously specified.

```
"scenario": {
  "ego": {
    "x": <random min="-3" max="3"/>,
    "v": <choice>
          <text>0</text>
          <random min="0" max="30"/>
          <random min="30" max="100"/>
         </choice> },
  "obstacles": [
    { "x": <choice>
            <random min="-50" max="-3"/>
            <random min="3" max="50"/>
           </choice>,
      "v": <choice>
            <random min="-100" max="0"/>
            <text>0</text>
            <random min="0" max="100"/>
           </choice> }, ...
  ]
}
```

```
ego_v: stop, slow, fast
obstacles_1_x: back, front
obstacles_1_v: backward, stop, forward
obstacles_2_x: back, front
obstacles_2_v: backward, stop, forward
```

Fig. 3. Above: A “scenario” with discrete choices, where “ego” is either stopping, moving slow, or moving fast, each of the two obstacles is either in front or back, and either stopping, moving forward or moving backward. Below: A corresponding PICT model, with informative names (in the implementation they get non-informative names).

as this is a prototype, and we use JSON as an example (Fig. 2). One can also think of a well-defined XML provided that tag names defined in our language do not conflict. The choice of real values is done by random testing; thus one may sample many concrete scenarios out of one $\ell 1$ script. In this preliminary work we leave it to domain experts to judge how many scenario should be sampled form an $\ell 1$ script.

Layer 2: This layer allows discrete <choice> tags. In practice, a <choice> tag should have at least two children, where each child represents an option for the choice. These choices are resolved either by random testing or combinatorial testing. For instance, Fig. 3 shows an $\ell 2$-script and a corresponding PICT model.

Layer 3: This final layer allows <repeat> tags, which indicate that the contents can occur arbitrarily many times, where attributes minOccurs and maxOccurs specify the range of the number of occurrences. If the delim attribute is specified, then its value is put in between repetitions. Here again a domain knowledge is (yet) necessary to argue how

```
"scenario": {
  "ego": {
    "x": <random min="-3" max="3"/>,
    "v": <choice>
          <text>0</text>
          <random min="0" max="30"/>
          <random min="30" max="100"/>
         </choice> },
  "obstacles": [
   <repeat minOccurs="1" maxOccurs="5" delim=",">
    { "x": <choice>
            <random min="-50" max="-3"/>
            <random min="3" max="50"/>
           </choice>,
      "v": <choice>
            <random min="-100" max="0"/>
            <text>0</text>
            <random min="0" max="100"/>
           </choice> }
   </repeat>
  ]
}
```

Fig. 4. A fully generalized “scenario”, with at most five obstacles.

many different numbers of repetition should be tested.

III. Conclusion and Future Work

In this extended abstract, we proposed a language to describe *scenarios* where both continuous real-valued parameters and discrete parameters are involved. We provide a prototype implementation of a tool that inputs such a scenario description and outputs a set of concrete scenarios by combining random testing and combinatorial testing. This preliminary work opens up a vast field for further exploration.

- How effective is combinatorial testing for cyber physical systems in terms of failure detection?
- What kind of combinatorial coverage should we focus on? For instance, it seems redundant to distinguish obstacle 1 and obstacle 2 in our example. Also when considering a scenario with *time stages*, intuitively we would like to consider a *mixed-strength* coverage concerning only combinations of adjacent time stages.
- How do we handle *constraints* which, on cyber physical systems, are naturally about real variables and real functions. It is nontrivial how to turn real constraints into discrete constraints on the combinatorial testing level.
- How should we systematically sample from $\ell 1$ and $\ell 3$ scripts? We plan to incorporate *adaptive random testing* [2] and *regular language sampling* [1].

References

[1] P. Arcaini, A. Gargantini, and E. Riccobene. Fault-based test generation for regular expressions by mutation. *Software Testing, Verification and Reliability*, March 2018.

[2] Tsong Yueh Chen, Fei-Ching Kuo, Robert G. Merkel, and T. H. Tse. Adaptive random testing: The ART of test case diversity. *Journal of Systems and Software*, 83(1):60–66, 2010.

[3] Jacek Czerwonka. Pairwise testing in real world. In *PNSQC 2006*, pages 419–430, 2006.

[4] Richard Hamlet. *Random Testing*. American Cancer Society, 2002.

[5] D. Richard Kuhn, Raghu N. Kacker, and Yu Lei. *Introduction to Combinatorial Testing*. CRC press, 2013.