A Visual Logical Language for System Modelling in Combinatorial Test Design

Maria Spichkova[1], Anna Zamansky[2(✉)], and Eitan Farchi[3]

[1] RMIT University, Melbourne, Australia
maria.spichkova@rmit.edu.au
[2] University of Haifa, Haifa, Israel
annazam@is.haifa.ac.il
[3] IBM Haifa Research Lab, Haifa, Israel

Abstract. This position paper addresses some weaknesses of the standard logical languages used for specification of system models in combinatorial test design. To overcome these weaknesses, we propose a new logical language which uses visual elements with the aim to lower the cognitive load of the modeller and thereby reduce the risk of modelling errors.

Keywords: Cognitive aspects of modelling · Combinatorial test design

1 Introduction

Combinatorial Test Design (CTD) is an effective methodology for test design of complex software systems, in which a system is modelled using a *combinatorial model* [6,16]. The CTD approach aims to systematically optimise the number of test cases, while ensuring the coverage of given conditions.

There is a large body of works on different aspects of CTD (see [6] for a comprehensive survey), and numerous tools have been developed (e.g., 40 tools are currently listed in [7]). However, according to [6], only 5 % of publications on CTD address the process of constructing combinatorial models. As noted in [13], *"An under-explored challenge for wide deployment of CTD in industry is the manual process for modelling and maintaining the test space"*. Indeed, the task of combinatorial model construction for complex systems heavily relies on tacit human knowledge, and therefore will always remain the task of a human tester. The complexity and error-prone nature of this task calls for more emphasis on human-centric approaches for supporting CTD.

In previous works we have considered two directions for more human-centric support of the manual process of combinatorial model construction: agile error correction [15] and support of modelling with multiple levels of abstraction [11, 12]. In this paper, we focus on the *logical language* for specifying the restrictions in combinatorial models and discuss how it can be modified to better serve the needs of the human modeller.

J. Krogstie et al. (Eds.): CAiSE 2016, LNBIP 249, pp. 116–121, 2016.
DOI: 10.1007/978-3-319-39564-7_12

The logical languages currently used in CTD tools are based on the standard Boolean semantics. This semantics has been criticised as inadequate for supporting model evolution in [13], where a more sophisticated lattice-based semantics was proposed. In this position paper we point out some implicit assumptions which are made when using Boolean semantics in the context of CTD. We propose simple visual constructs to make these assumptions more explicit, thereby reducing the cognitive load of the modeller when specifying the logical restrictions, as well as the chances for human errors. We show how these constructs can be used as an alternative solution to the problems pointed out in [13].

2 Logical Restrictions in CTD Models

A *combinatorial model*, consists of a set of parameters, their respective possible values and a set of logical restrictions on the value combinations [6,16]. Thus, in CTD a system is modelled using a finite set of system parameters $\mathbf{A} = \{\mathbf{A}_1, \ldots, \mathbf{A}_n\}$ together with their corresponding associated values $\mathbf{V} = \{\mathbf{V}(\mathbf{A}_1), \ldots, \mathbf{V}(\mathbf{A}_n)\}$. A *scenario* (or test) is an assignment of a value from $\mathbf{V}(\mathbf{A}_i)$ to each $\mathbf{A}_i$. A *combinatorial model* for $\mathbf{A}$ is defined as a set of scenarios (tests).

For an example, consider the standard toy combinatorial model often used in CTD literature, cf. [13]. Suppose our system is modelled using three parameters:

- ItemStatus (denoted by **IS**)
- OrderShipping (denoted by **OS**)
- DeliveryTimeframe (denoted by **DT**)

Thus, the set of parameters is specified by $\mathbf{A} = \{\mathbf{IS}, \mathbf{OS}, \mathbf{DT}\}$, where the corresponding values of the parameters are defined by

$$\begin{aligned}\mathbf{IS} &= \{InStock, OutOfStock, NoSuchProduct\}\\ \mathbf{OS} &= \{Air, Ground\}\\ \mathbf{DT} &= \{Immediate, 3\,Days, 1\,Month\}\end{aligned}$$

Assuming that there are only three parameters in this example, a combinatorial model of the system is a set of scenarios, (which are assignments of values to parameters), such as:

$$\begin{aligned}s_1 &: (\mathbf{IS} = InStock, \mathbf{OS} = Air, \mathbf{DT} = Immediate)\\ s_2 &: (\mathbf{IS} = InStock, \mathbf{OS} = Ground, \mathbf{DT} = Immediate)\end{aligned}$$

There are overall 18 possible scenarios in this example. However, in practice not all the scenarios are executable: e.g., it is not possible to have an immediate delivery time when the item is sent via ground. Therefore, the most challenging manual task in combinatorial modelling is separating the valid (executable) scenarios, and ruling out the invalid ones.

Usually this is done by using some dialect of classical logic. Figure 1 presents logical restrictions from a model for the above example, constructed using IBM FoCuS tool (IBM Functional Coverage Unified Solution), cf. [10,14].

Type	Expression
Exclusion	OrderDeliveryTimeframe.equals("oneMonth") && OrderShipping.equals("air")
If-Then	[OrderDeliveryTimeframe == "immediate"] ==> [OrderShipping != "ground"]

Fig. 1. Logical restrictions in IBM FoCuS

The semantics of the logical languages used in the context of combinatorial modelling is (implicitly) assumed to be the standard Boolean semantics. As we explain below, this implicit assumption may cause ambiguity and confusion at the time of model construction. In [13] the Boolean semantics is also criticised as inadequate for supporting model evolution. One of the main criticisms is an inconsistent interpretation of test validity in case a new value is added. Considering, e.g., the following logical restriction which can used to specify a combinatorial model: $R_1 : \mathbf{DT} = Immediate \rightarrow \mathbf{OS} \neq Ground$.

If a new value, e.g., *Sea*, is added to $\mathbf{V}(\mathbf{OS})$, then the test (scenario) $s_3 = (\mathbf{IS} = InStock, \mathbf{OS} = Sea, \mathbf{DT} = Immediate)$ is valid, according to R_1. Now the following restriction is equivalent to R_1 in the sense that it induces the same set of valid tests: $R_2 : \mathbf{DT} = Immediate \rightarrow \mathbf{OS} = Air$. But using R_2 instead of R_1 renders s_3 invalid, leading to an inconsistent interpretation of tests including the new value *Sea*.

Tzoref-Brill and Maoz [13] propose an alternative, lattice-based semantics to deal with the above problem, as well as another problem related to splitting values of a parameter. Referring to alternative solutions, they note: *"One may suggest to remove the negation operator from the constraint language made available to the practitioner. While this will resolve the inconsistent interpretation, it will extremely limit the flexibility of the practitioner to specify constraints in a concise manner, and is thus infeasible in practice"*.

We believe that the problematic negation operator is a symptom of a much deeper problem: lack of explicit representation of extra assumptions made in the CTD domain. The logic of CTD is related to classical logic in roughly the same way as in database theory: quite similar, but the devil is in the details. More concretely, a standard approach in database theory assumes the closed world assumption (CWA), cf. [8], according to which whatever is not entailed by the database is assumed to be false. A formalisation of this assumption, therefore, consists in adding to the database the negations of all literals not entailed by it. Another important issue in database theory is that of *integrity constraints* [9]. The idea is that only certain database states are considered acceptable, and an integrity constraint is meant to enforce these legal states.

Assumptions close in spirit to the above are mirrored in the CTD domain. The first is that whatever tests are not excluded by the logical restrictions, are considered valid. The second is that each parameter $\mathbf{A}$ of the model may assume *exactly* one of its possible values $\mathbf{V}(\mathbf{A})$.

We can now note the following: the two logical restrictions R_1 and R_2 are *not* logically equivalent in first-order classical logic (with equality). They are equivalent under the explicit integrity constraint $IC_1 : \mathbf{OS} = Air \vee \mathbf{OS} = Ground$. Note also that after adding the new value *Sea* to $\mathbf{V}(\mathbf{OS})$, the integrity constraint IC_1

becomes invalid. Hence, the answer to the question whether the test s_3 is valid depends on the assumed integrity constraint. Under the constraint IC_1 above, it is invalid. Under the refined constraint $IC_2 : \mathbf{OS} = Air \vee \mathbf{OS} = Ground \vee \mathbf{OS} = Sea$. it is valid. Therefore, what is called in [13] inconsistent interpretation, can be viewed as merely a change in the assumed integrity constraints.

3 A New Visual Logical Language

In light of the problems discussed above, our aim is to make the logical language of CTD modelling less ambiguous by integrating *explicitly* the assumption that any parameter $\mathbf{A}$ assumes *exactly* one value from $\mathbf{V}(\mathbf{A})$. In other words, given $\mathbf{V}(\mathbf{A}) = \{v_1, \ldots, v_n\}$, it holds that $\mathbf{A} = v_1 \vee \mathbf{A} = v_2 \vee \ldots \mathbf{A} = v_n$. Thus, if $\mathbf{A} \neq v_i$, then the value of $\mathbf{A}$ belongs to the set $\{v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n\}$. Drawing inspiration from the syntax of labelled formulas in non-classical logics (cf. [1,2]), we think of the basic atomic formula in the proposed language as an expression of the form: $\mathbf{A} : L$, where $L \subseteq \mathbf{V}(\mathbf{A})$. This formula is supposed to be true (or to hold) iff the value of $\mathbf{A}$ is an element of the set L.

The visual element corresponding to $\mathbf{A} : L$ explicitly contains both L and $\mathbf{V}(\mathbf{A})\backslash L$. More concretely, any such element is represented a partition of $\mathbf{V}(\mathbf{A})$ into valid and invalid elements, to remind the tester *explicitly* about the assumptions discussed above, cf. Fig. 2.

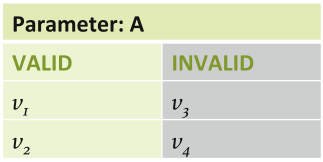

Fig. 2. Basic partition formula for a parameter $\mathbf{A}$ (Color figure online)

The above visual formula is equivalent to the Boolean formula $(\mathbf{A} = v_1 \vee \mathbf{A} = v_2 \vee \mathbf{A} = v_3 \vee \mathbf{A} = v_4) \wedge \mathbf{A} \neq v_3 \wedge \mathbf{A} \neq v_4$, and so also entails $\mathbf{A} = v_1 \vee \mathbf{A} = v_2$.

It is important to note that negation is built-in already, by using different colours to denote the valid and invalid values. This resolves the problematic issues with using negation discussed above. This is not accidental that we used light green colour to denote valid values, and grey to denote invalid values. This conforms to the common standards in interface design, cf. [4]: grey is often used to denote options that exist but are not accessible for some reasons. Moreover, we would like to avoid combinations like green/red, which are perceived as aggressive by many users, and, moreover, are not distinguishable by colour-blind users. We also aim to avoid having areas of strong colour and high contrast, as they can produce afterimages when the viewer looks away from the screen, which increases the cognitive load and visual stress from prolonged viewing [3]. On

addition to the (implicit) negation, the basic elements (or partitions) such as the one demonstrated on Fig. 2 can be connected via the usual binary logical operators of implication, conjunction and disjunction, whose semantics remains standard (and self-explanatory). Figure 3 shows that, e.g., the logical restrictions R_1 and R_2 described above have the same representation in our visual language:

Fig. 3. Representation of the logical conditions R_1/R_2

Note that our visual language provides simple and intuitive support for handling the problems of refinement and splitting, discussed in [13]. For instance, adding a new value 'Sea' to the Order Shipping (**OS**) parameter values should enforce the modeller to make a decision where to place it in the formula: either in the valid or invalid zone, see Fig. 4:

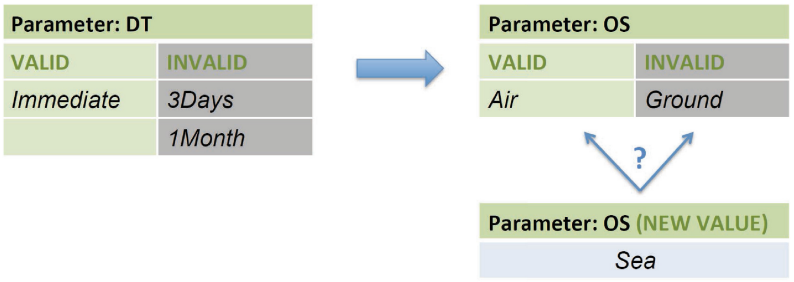

Fig. 4. Refining the logical condition when a new value is added

4 Summary and Future Research

The need for human-centric approaches in supporting the process of combinatorial test design is increasingly acknowledged [6,13]. This paper explores how standard logical languages used for the specification of model constraints can be better adapted to the needs of the human modeller. We propose a variation of a classical logic dialiect, in which implicit assumptions made in the context of CTD are explicitly represented using visual elements. Our next step will be implementing and evaluating a tool based on the principles proposed in this paper. We also plan to extend this work to support model evolution [13], update [10] and refinement [11]. Another interesting direction is investigating the ways in which human-centric approaches of combinatorial test design can benefit from the large body of research on cognitive effectiveness of visual notations, cf. [5].

References

1. Baaz, M., Fermüller, C., Salzer, G., Zach, R.: Labeled calculi and finite-valued logics. Stud. Logica. **61**(1), 7–33 (1998)
2. Baaz, M., Lahav, O., Zamansky, A.: Finite-valued semantics for canonical labelled calculi. J. Autom. Reasoning **51**(4), 401–430 (2013)
3. MacDonald, L.: Using color effectively in computer graphics. IEEE Comput. Graph. Appl. **19**(4), 20–35 (1999)
4. Marcus, A.: Siggraph'93 tutorial notes: graphic design for user interfaces (1993)
5. Moody, D.: The "physics" of notations: toward a scientific basis for constructing visual notations in software engineering. IEEE Trans. Softw. Eng. **35**(6), 756–779 (2009)
6. Nie, C., Leung, H.: A survey of combinatorial testing. ACM Comput. Surv. (CSUR) **43**(2), 11 (2011)
7. Pairwise Testing Website. http://www.pairwise.org/tools.asp
8. Reiter, R.: On closed world data bases. In: Gallaire, H., Minker, J. (eds.) Logic and Data Bases, pp. 55–76. Springer, Heidelberg (1978)
9. Reiter, R.: On integrity constraints. In: Proceedings of the 2nd Conference on Theoretical Aspects of Reasoning about Knowledge, pp. 97–111. Morgan Kaufmann Publishers Inc. (1988)
10. Segall, I., Tzoref-Brill, R.: Interactive refinement of combinatorial test plans. In: 2012 34th International Conference on Software Engineering (ICSE), pp. 1371–1374 (2012)
11. Spichkova, M., Zamansky, A.: A human-centred framework for combinatorial test design. In: Proceedings of ENASE (2016)
12. Spichkova, M., Zamansky, A., Farchi, E.: Towards a human-centred approach in modelling and testing of cyber-physical systems. In: Proceedings of the International Workshop on Automated Testing of Cyber-Physical Systems in the Cloud (2015)
13. Tzoref-Brill, R., Maoz, S.: Lattice-based semantics for combinatorial model evolution. In: Finkbeiner, B., et al. (eds.) ATVA 2015. LNCS, vol. 9364, pp. 276–292. Springer, Heidelberg (2015). doi:10.1007/978-3-319-24953-7_22
14. Wojciak, P., Tzoref-Brill, R.: System level combinatorial testing in practice-the concurrent maintenance case study. In: Proceedings of the 2014 IEEE International Conference on Software Testing, Verification, and Validation, pp. 103–112. IEEE Computer Society (2014)
15. Zamansky, A., Farchi, E.: Helping the tester get it right: towards supporting agile combinatorial test design. In: 2nd Human-Oriented Formal Methods workshop (HOFM 2015) (2015)
16. Zhang, J., Zhang, Z., Ma, F.: Introduction to combinatorial testing. In: Zhang, J., Zhang, Z., Ma, F. (eds.) Automatic Generation of Combinatorial Test Data. SpringerBriefs in Computer Science, pp. 1–16. Springer, Heidelberg (2014)