Consistency Checking of Functional Requirements

Simone Vuotto

[1] Università degli Studi di Genova
simone.vuotto@edu.unige.it
[2] Università degli Studi di Sassari
svuotto@uniss.it

Abstract. Requirements are informal and semi-formal descriptions of the expected behavior of a system. They are usually expressed in the form of natural language sentences and checked for errors manually, *e.g.*, by peer reviews. However, manual checks are error-prone and time-consuming. With the increasing complexity of cyber-physical systems and the need of operating in safety- and security-critical environments, it became essential to automatize the consistency check of requirements and build artifacts to help system engineers in the design process.

1 Introduction

The assessment of requirements is an important yet costly and complex task, still largely carried out manually. The Requirements Engineering (RE)[8] research field aims at developing tools and techniques to analyze and handle requirements in a more efficient and automatic way. One of the main challenges is to evaluate requirements *consistency*: informally, it means detecting errors, missing information and deficiencies that can compromise the interpretation and implementation of the intended system behavior. At a syntactic level, this may involve the check for compliance with standards and guidelines, such as the use of a restricted grammar and vocabulary. We call this task *Compliance Checking*.

However, most of the inconsistencies reside at a semantic level, *i.e.* in their intended meaning. This call for an interpretation and reasoning of requirements semantics. An open and interesting research question is how to formalize and translate requirements into a formal representation. A recurrent solution in the literature is the use of Property Specification Patterns (PSPs), first introduced by [3]. PSPs provide a direct mapping from English-like structured natural languages to one or more logics. A survey of all available patterns and their translation has been made by [2]. Other approaches, like [4], employ Natural Language Processing techniques to extract the representation directly from fully natural language requirements.

Given the set of requirements represented in a formal logic, another fundamental question is what kind of reasoning we can employ and how to do that. We formally define this task *Consistency Checking* analysis [5]. Consistency Checking can range from simple variables type and domain checks to more complex

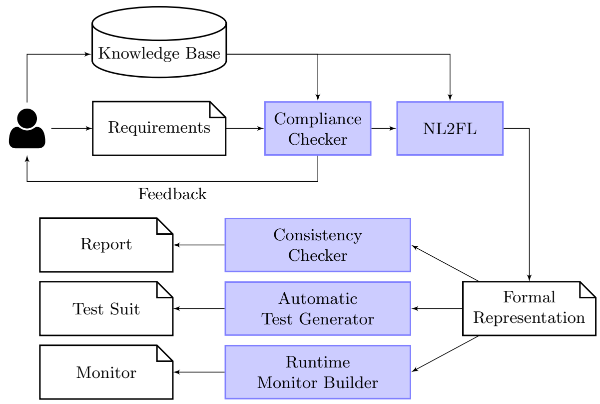

Fig. 1. General framework of the requirement analysis tool

activities, like the evaluation of the intended system behavior over time. In particular, we are interested in checking if the set of requirements together "make sense", namely answering the question:

Given the set of requirements, does a system exist that can satisfy them all at the same time?

The choice of which logic to use largely affects the reasoning power and the kind of requirements that can be formalized: qualitative, real-time and/or probabilistic. For example, in some logics it is only possible to specify that an event e will eventually happen in the future, while others can also constraint the time frame (*e.g.* e will happen within 5s) or its likelihood (*e.g.* e will happen with probability $p \geq 0.5$). We started from Linear Temporal Logic (LTL)[9] because it has a good balance between expressiveness and complexity, and it is widely used in the literature. In particular, answering the aforementioned question can be easily translated in a LTL satisfiability check, largely studied and with many efficient tools available [10].

The satisfiability check in turn brings other two research questions:

- *Vacuity Check* [6]: if the formula is satisfiable, is it satisfiable in an interesting way? For example, the LTL specification $\Box(msg \rightarrow \Diamond rcv)$ ("every message is eventually received") is satisfied in a model with no messages, but it is possibly not the expected behavior. Being able to pinpoint these situations can give the user interesting information to modify or expand the requirements document.
- *Inconsistent Requirements Explanation*: if the requirements are inconsistent, which is the minimum set of them that create the inconsistency? The number of requirements may be really large, but only few of them making the system unfeasible.

Finally, the formalization of requirements and the consistency checking are enablers for other tasks we would like to tackle in this Ph.D. project, namely the automatic generation of test suites and runtime monitors. The full overview of the tool that we are designing is depicted in Figure 1. We are now focusing on the NL2FL and Consistency Checker modules.

2 Consistency of Property Specification Patterns

Our first contribution [7], developed in the context of the H2020 CERBERO European Project [1], presented a tool for the consistency checking of qualitative requirements expressed in form of PSPs with constrained numerical signals. An example of requirement that we can handle is:

Globally, it is always the case that if proximity_sensor < 20 *holds, then arm_idle eventually holds.*

We first translate every requirement $r_i \in R$ in LTL($\mathcal{D}_C$), an extension of LTL over a constraint system $D_C = (\mathbb{R},<,=)$, with atomic constraints of the form $x < c$ and $x = c$ (where $c \in \mathbb{R}$ is a constant real number and '<" and "=" have the usual interpretation). We then show how the new problem can be reduced to LTL satisfiability. Let $X(\phi)$ be the set of numerical variables and $C(\phi)$ be the set of constants that occur in ϕ. We compute:

- the LTL($\mathcal{D}_C$) formula ϕ_i for every requirement $r_i \in R$;
- the conjunctive formula $\phi = \phi_1 \wedge ... \wedge \phi_n$;
- a set $M_x(\phi)$ of boolean propositions representing possible values of $x \in X(\phi)$;
- the formula Q_M encoding the constraints over $M_x(\phi)$ $\forall x \in X(\phi)$;
- the formula ϕ' that substitute all $x \in X(\phi)$ in ϕ with a set of boolean propositions from $M_x(\phi)$;

Given the LTL($\mathcal{D}_C$) formula ϕ over the set of Boolean atoms *Prop* and the terms $C(\phi) \cup X(\phi)$ we have that ϕ is satisfiable if and only if the LTL formula $\phi_M \rightarrow \phi'$ is satisfiable. This result is important because it shows that LTL($\mathcal{D}_C$) is decidable and that we can exploit state-of-the-art LTL model checkers.

3 Future work

In order to reduce the number of errors in the specification, we have partially implemented an algorithm to check the relationship among requirements. This is a first step to prevent vacuous results, but more work is needed.

Connected Requirements Check Given a set of requirements $R = \{r_1, ..., r_n\}$, we want to check if one or more of such are completely unrelated to the others, meaning that they describe some behaviors that do not interact with the main bulk of the system. This may happen in an underspecified requirements set or for some spelling errors. To find these faulty requirements we first build the undirected graph $G = (V, E)$ representing the connections in R, such that:

- $v_i \in V \ \forall r_i \in R$;
- $(i, j) \in E$ if $X(r_i) \cap X(r_j) \neq \emptyset \ \forall r_i, r_j \in R, i \neq j$

where $X(r_i)$ is set of variables, boolean or numerical, that appear in r_i. We then compute all the connected components in G. If the number of components in greater than one, we find the smallest one (*i.e.* the component with the lowest number of vertex) and report it to the user.

Currently we are also focusing our attention on the Inconsistent Requirements Explanation problem. We implemented a simple algorithm that iterate over all $r_i \in R$ and perform the consistency check on the set $R \setminus r_i$. We keep r_i in R only if the new set is shown to be consistent, and we discard it otherwise. The algorithm terminates when all the requirements in the original set are checked. This algorithm effectively find a solution, but it is quite inefficient. Therefore, we are seeking for a better algorithm which exploits the structure of the problem.

Finally, for future works we would also like to both extend the natural language interface with less restrictive constraints and adopt a more expressive logic. In particular, we are interested in probabilistic logics such PCTL, but the consistency checking problem is difficult to define in this case and more research is needed.

Acknowledgments The research of Simone Vuotto is funded by the EU Commission H2020 Programme under grant agreement N.732105 (CERBERO project).

References

1. H2020 Cerbero Project Website. http://www.cerbero-h2020.eu/
2. Autili, M., Grunske, L., Lumpe, M., Pelliccione, P., Tang, A.: Aligning qualitative, real-time, and probabilistic property specification patterns using a structured english grammar. IEEE Transactions on Software Engineering 41(7), 620–638 (2015)
3. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: Proceedings of the 21st International conference on Software engineering. pp. 411–420 (1999)
4. Ghosh, S., Elenius, D., Li, W., Lincoln, P., Shankar, N., Steiner, W.: Arsenal: automatic requirements specification extraction from natural language. In: NASA Formal Methods Symposium. pp. 41–46. Springer (2016)
5. Heitmeyer, C.L., Jeffords, R.D., Labaw, B.G.: Automated consistency checking of requirements specifications. ACM Transactions on Software Engineering and Methodology (TOSEM) 5(3), 231–261 (1996)
6. Kupferman, O., Vardi, M.Y.: Vacuity detection in temporal model checking. International Journal on Software Tools for Technology Transfer 4(2), 224–233 (2003)
7. Narizzano, M., Pulina, L., Tacchella, A., Vuotto, S.: Consistency of property specification patterns with boolean and constrained numerical signals. In: NASA Formal Methods Symposium. pp. 383–398. Springer (2018)
8. Nuseibeh, B., Easterbrook, S.: Requirements engineering: a roadmap. In: Proceedings of the Conference on the Future of Software Engineering. pp. 35–46. ACM (2000)
9. Pnueli, A.: The temporal logic of programs. In: Foundations of Computer Science, 1977., 18th Annual Symposium on. pp. 46–57. IEEE (1977)
10. Rozier, K.Y., Vardi, M.Y.: Ltl satisfiability checking. International journal on software tools for technology transfer 12(2), 123–137 (2010)