Syntactic Theory Functors

Magne Haveraaen[1] and Markus Roggenbach[2]

[1] Bergen Language Design Laboratory
Department of Computer Science, University of Bergen, Norway
https://bldl.ii.uib.no/
[2] Department of Computer Science, Swansea University, Wales, UK
http://www.cs.swan.ac.uk/~csmarkus/

Algebraic specification has proven to be a powerful mechanism in software engineering. Examples include the formulation of concise requirements, analysing requirements for consistency [6], automated random testing for quality assurance [1,2], and the verification of software designs [4].

Like in programming, where one can distinguish between programming 'in the large' and 'programming in the small' [3], also in algebraic specification one distinguishes between basic and structured specification [5]. Structuring specification is considered 'good practice' for many reasons including separation of concerns, ease of reuse of specification-text, and improved theorem proving support. In particular, the algebraic specification language CASL offers a wealth of structuring mechanisms, including renaming, extension, union, hiding, and parameterisation. However, developing specifications for practical problems still hits issues of structuring and reusing specifications. Though immaterial to foundational specification theory, lack of support causes lengthy writing of boilerplate code or repeated adaptation of specifications from one context to another.

Thus, we suggest to go a step beyond the typical structuring mechanism. Concretely, we suggest *syntactic theory functors* (STFs) as a means to 'produce' complex specifications from simpler ones. An STF is a functor F from specifications to specifications, such that

- the application of F to a given specification Sp, written as $F(Sp)$, can be flattened out into a basic specification, i.e., one can represent it in the language used to write Sp and therefore understand the effect of F from the flattened form; and
- F is compatible with the basic structuring mechanisms.

STFs opens up new ways of reusing existing and structuring new specifications by enlarging the collection of structuring mechanisms available to a specification developer. Using STFs will simplify maintenance of specifications compared to the copy-paste-adapt which otherwise is needed. An STF application can stand as a shorthand for a specification that would be 'hard' and 'lengthy' to write directly, e.g., in CASL. We clearly see the need to use such functors in application areas such as partial differential equations, volume graphics, compilers/transformations, etc.

In our paper we provide a selection of useful theory functors, demonstrate their application in several examples, and study their structural properties.

References

1. Bagge, A.H., David, V., Haveraaen, M.: Testing with axioms in C++ 2011. Journal of Object Technology 10, 10:1–32 (2011), https://doi.org/10.5381/jot.2011.10.1.a10
2. Crispim, P., Lopes, A., Vasconcelos, V.T.: Runtime verification for generic classes with ConGu 2. In: Davies, J., Silva, L., da Silva Simão, A. (eds.) Formal Methods: Foundations and Applications - 13th Brazilian Symposium on Formal Methods, SBMF 2010, Natal, Brazil, November 8-11, 2010, Revised Selected Papers. Lecture Notes in Computer Science, vol. 6527, pp. 33–48. Springer (2010), https://doi.org/10.1007/978-3-642-19829-8_3
3. DeRemer, F., Kron, H.H.: Programming-in-the-large versus programming-in-the-small. IEEE Trans. Software Eng. 2(2), 80–86 (1976), https://doi.org/10.1109/TSE.1976.233534
4. James, P.: SAT-based Model Checking and its applications to Train Control Software. Master's thesis, Swansea University (2010)
5. Mosses, P.D. (ed.): CASL Reference Manual: The Complete Documentation of the Common Algebraic Specification Language, Lecture Notes in Computer Science, vol. 2960. Springer (2004), https://doi.org/10.1007/b96103
6. Roggenbach, M., Schröder, L.: Towards trustworthy specifications I: Consistency checks. In: Cerioli, M., Reggio, G. (eds.) Recent Trends in Algebraic Specification Techniques, 15th International Workshop, WADT 2001. Lecture Notes in Computer Science, vol. 2267. Springer (2001), https://doi.org/10.1007/3-540-45645-7_15