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


