we have defined a formal semantics of *MiniMaple* annotated programs [3] to formally describe the runtime behavior of such computer algebra programs. Currently we are developing a verification calculus where we need to translate annotated *MiniMaple* into a semantically equivalent Why3 [12] (an intermediate verification framework) program. Also the verification conditions generated by the calculus must be sound with respect to the semantics.

Computer algebra programs written in an annotated *MiniMaple* are semantically more complex than classical and functional programming and specification languages as they are fundamentally different from these languages. As a consequence, the denotational semantics of *MiniMaple* and its annotations which we have developed have the following features:

- The denotational semantics is defined as a state relationship between pre- and post-states.
- *MiniMaple* has expressions with side-effects, which is not supported in functional programming languages, e.g. Haskell [9] and Miranda [10]. As a result the evaluation of an expression may change the program execution state.

The research was funded by the Austrian Science Fund (FWF): W1214-N15, project DK10.

further details of the formal syntax of *MiniMaple* and its specification language, please see [1], [2].

The rest of the paper is organized as follows: Section 2 gives state of the art of formal semantics of programming and specification languages. In Section 3, we discuss the overview of the denotational semantics of *MiniMaple* and its specification language. Sections 4 defines the semantics of *MiniMaple* and Sections 5 and 6 give the definitions of the semantics of specification expression and annotation languages respectively. Section 7 presents conclusions and future work.

II. State of the Art

In this section we first sketch a state of the art of various approaches of defining formal semantics of classical and functional programming languages and specification languages and then discuss their differences to the computer algebra programming and specification languages.

The main approaches of defining formal semantics are operational, denotational and axiomatic [24]. The denotational method is widely used as it defines the meaning of a program as a mathematical function that maps abstract syntax to its semantic value, a denotation. Usually this function is partial because of the fact that some syntactically correct programs may not have defined meanings. Since the denotational method

978-0-7695-4927-9/12 $26.00 © 2012 IEEE
DOI 10.1109/FIT.2012.38