Our source program analyzer does name analysis and most of the needed error checking on our input. We have two different kinds of variables to store in the definition table: grammar symbols (which are similar to function definitions in some ways) and parameters.

The appearance of a grammar symbol (like A or Trunk) on the left hand side of an arrow is considered a defining occurrence, and only one defining occurrence of each grammar symbol is allowed. The scope of the symbol is the entire “Lsys” scope. Because of this, left-hand sides of the arrows must be evaluated before any right-hand sides. In addition, like functions, grammar symbols have some number of parameters (at least two). Any use of a grammar symbol on the right hand side of an arrow must have the same number of parameters as when it was defined. However, since our specification language has no types, we chose to simply write this computation rather than using the built-in function-checking computations. The only property of a grammar symbol is this number of arguments count. Our analyzer checks for and reports the following errors:

- A grammar symbol is defined multiple times
- A grammar symbol is used but never defined
- A grammar symbol is used with the wrong number of parameters

Our other variable type is a parameter. Here the scoping rules are more complicated. Any instance of a parameter in a rule context (on either side of the arrow) is considered a defining occurrence. If it is on the left-hand side of the arrow, it is defined for the scope of the rule. If it is on the right-hand side and not already defined, it is defined in the “Lsys” scope (not the rule scope). This required us to write a custom C binding function.

In addition, after the rules have been defined a list of details about each parameter appears in the specification. Parameter names here are considered applied occurrences, and are only valid if they have been defined in the “Lsys” scope by the grammar. The properties defined by these listings are treated as properties of the parameters (stored as a struct since there are several of them). These properties may only be defined for each parameter once. If a “ticks” property is given, the parameter must previously have been given a “continuous” property. The final analysis done on parameters is to detect which Symbol classes in the output will need access to each parameter. Three types of symbols (axiom, move, and angle) are tracked; if only one symbol needs a parameter then its key is also needed and stored (with the exception of the axiom, since there is only one axiom). Finally, a known constant parameter called “Number of iterations” is defined in the “Lsys” scope automatically, and is known to be used in the axiom symbol.

The following errors and potential errors are detected for parameters:

- A warning is issued if no attributes are defined for some “Lsys” scope parameter (including “Number of iterations”)
- An error is issued if attributes are defined for a parameter that does not exist in the “Lsys” scope
- An error is issued if the user attempts to give a parameter properties (continuous or discrete) multiple times
• An error is issued if the user attempts to give “ticks” to a parameter not previously declared continuous
• Errors are issued if the given “min”, “max”, and “initial values” of a parameter do not make sense, such as if “min” is greater than “max”

With the exception of binding certain parameters, all of these computations and checks use the standard name analysis module with C scoping rules and standard LIDO computations (primarily IF statements to detect errors).