Abstract Syntax Tree Rationale

Criteria:
My main goal for the abstract syntax tree is that it makes the translator as easy as possible to write and to debug, but still remains useful for the user. I turned that goal into the following three criteria used to rate my abstract syntax tree, in descending order of importance:

1. Allow for name analysis: the tree should be structured in a way that allows name analysis to be performed on the user-supplied expressions. If the translator can’t do name analysis on these then it will be left to the C compiler. This would require the user to interpret C compiler errors, which is not desirable. Importance: 100.
2. Make output as easy as possible: the output from the translator will have a very different structure than the input, so small changes in tree design could have a big impact on how easy it is to generate the output. Importance: 80.
3. Easy to see correctness: it should be easy to tell if a tree is correct by looking at a graphical representation. This is accomplished by ensuring there are no extraneous rules, and by arranging the rules in a logical order. This will make it easier to debug the AST, but not necessarily easier to debug rule computations. Importance: 50.

Score:
I have scored my AST according to the above criteria as follows:

1. The user-supplied expressions could have been stored as strings instead of parsed into the AST, but this would have precluded doing any name analysis on them. The score for this criterion is not perfect because the expressions allowed in are fairly limited. Score: 90/100.
2. Output is tricky is because the value of an identifier is not just a number, but an entire expression that will need to be inserted into the output in various places. To accomplish this I have defined a property Val of type PTGNode. When an identifier is declared, the output PTGNode for that identifier’s expression will be assigned to the identifier’s definition table key, so that it can be looked up later when the identifier is used. It would be much easier to simply store the expression as a string, but I wanted to be able to do name analysis on the expressions. The score for this criterion will be 100 minus the number of PTG computations I anticipate having to write (23, vs. 13 for the easier version). Score: 77.
3. For the most part there are no extraneous rules making the tree harder to understand. The operator (+, -, *, /) rules could be combined, but that would not make it any easier to verify an AST is correct for a certain input. All the different Layer rules make the AST slightly more complicated than it could be, but they are necessary for the output. Score: 90/100.

**Conclusions:**

My criteria forced me to make trade-offs between making the tool more useful for end-users and more difficult for me to write. I think I have made the correct trade-offs so that I will be able to finish the project in a short time frame, and still have it be a useful tool.

**Abstract Syntax Tree:**

\[
\begin{align*}
\text{ATTR Name: int;} \\
\text{ATTR Sym: int;} \\
\text{RULE Root: Network ::= Params Decls Layers END;} \\
\text{RULE rParams: Params ::= 'PARAMS' ParamList ';' END;} \\
\text{RULE rPrmList: ParamList LISTOF Param END;} \\
\text{RULE rParam: Param ::= VarDecl END;} \\
\text{RULE rDecls: Decls LISTOF Decl END;} \\
\text{RULE rDecl: Decl ::= VarDecl ':' Expr ';' END;} \\
\text{RULE rFuncDecl: Decl ::= 'FUNCTION' VarDecl ':' FileName ';' END;} \\
\text{RULE rVarDecl: VarDecl ::= Ident END;} \\
\text{RULE rFileName: FileName ::= String END;} \\
\text{RULE rLayers: Layers LISTOF Layer END;} \\
\text{RULE rOutLayer: Layer ::= 'OUTPUT' LayerSpec END;} \\
\text{RULE rInLayer: Layer ::= 'INPUT' LayerSpec END;} \\
\text{RULE rHiddenLyr: Layer ::= LayerSpec END;} \\
\text{RULE rLayerSpec: LayerSpec ::= 'LAYER' LayerName LayerDef 'ENDLAYER' END;} \\
\text{RULE rLayerName: LayerName ::= VarDecl END;} \\
\text{RULE rLayerDef: LayerDef ::= Decls END;} \\
\text{RULE rMultiply: Expr ::= Expr '*' Expr END;} \\
\text{RULE rAdd: Expr ::= Expr '+' Expr END;} \\
\text{RULE rSubtract: Expr ::= Expr '-' Expr END;} \\
\text{RULE rDivide: Expr ::= Expr '/' Expr END;} \\
\text{RULE rVarUseExp: Expr ::= VarUse END;} \\
\text{RULE rVarUse: VarUse ::= Ident END;} \\
\text{RULE rNumber: Expr ::= Number END;}
\end{align*}
\]
RULE rFuncCall: Expr ::= VarUse '(' Args ')' END;
RULE rArgs: Args LISTOF Expr END;
RULE rInteger: Number ::= Integer END;
RULE rFloat: Number ::= Float END;

CLASS SYMBOL IdentOcc COMPUTE SYNT.Sym = TERM; END;

SYMBOL LayerDef INHERITS RangeScope END;

SYMBOL VarDecl INHERITS IdDefScope, IdentOcc END;

SYMBOL VarUse INHERITS IdUseEnv, IdentOcc, ChkIdUse END;

**Concrete Syntax Tree:**

Expr: Term / Expr '+' Term / Expr '-' Term .
Term: Factor / Term '*' Factor / Term '/' Factor .
Factor: Number / VarUse / FuncCall .
FuncCall: VarUse '(' Args ')' .
Args: Expr* // ',' .