AST Grammar

December 19, 2015

1 Common

⟨*comment*⟩ ::= NullComment | (⟨*comment*⟩)

⟨*clock*⟩ ::= () | (ident [,ident]*) | (⟨*clock*⟩ [,⟨*clock*⟩]*)

⟨*integer*⟩ ::= [-+]?(0|[1-9][0-9]*)

⟨*float*⟩ ::= [-+]?(0|[1-9][0-9]*)(.[0-9]*[1-9])?

⟨*ident*⟩ ::= [a-zA-Z_][a-zA-Z0-9_]*

2 Program

⟨*topLevel*⟩ ::= TopLevel(⟨*mainBlk*⟩, ⟨*programBlk*⟩)

⟨*mainBlk*⟩ ::= main(⟨*ident*⟩)

⟨*programBlk*⟩ ::= program(⟨*stmtBlk*⟩ [, ⟨*stmtBlk*⟩]*)

⟨*stmtBlk*⟩ ::= ⟨*typeBlk*⟩ | ⟨*constBlk*⟩ | ⟨*nodeBlk*⟩

3 Type Block

⟨*typeBlk*⟩ ::= type_block(⟨*typeStmt*⟩ [, ⟨*typeStmt*⟩]*)

⟨*typeStmt*⟩ ::= type(⟨*ident*⟩, ⟨*type*⟩, ⟨*comment*⟩)

⟨*type*⟩ ::= bool | short | ushort | int | uint | float | real | char | ⟨*contruct*⟩ | ⟨*constructEnum*⟩
| ⟨*array*⟩ | ⟨*typename*⟩ | ([⟨*type*⟩ [,⟨*type*⟩]*])

⟨*construct*⟩ ::= construct(⟨*field*⟩ [, ⟨*field*⟩]*)

⟨*field*⟩ ::= field(⟨*ident*⟩, ⟨*type*⟩)

⟨*constructEnum*⟩ ::= construct_enum(⟨*ident*⟩ [, ⟨*ident*⟩]*)

⟨*array*⟩ ::= array(⟨*type*⟩, ⟨*intValue*⟩)

⟨*intValue*⟩ ::= INT(⟨*integer*⟩)

⟨*typename*⟩ ::= typename(⟨*ident*⟩)

4 Const Block

⟨*constBlk*⟩ ::= const_block(⟨*constStmt*⟩ [, ⟨*constStmt*⟩]*)

⟨*constStmt*⟩ ::= const(⟨*ident*⟩, ⟨*type*⟩, ⟨*value*⟩, ⟨*comment*⟩)

⟨*value*⟩ ::= ID(⟨*ident*⟩, ⟨*type*⟩) | BOOL(⟨*bool*⟩) | CHAR(⟨*integer*⟩) | SHORT(⟨*integer*⟩) | USHORT(⟨*integer*⟩) | INT(⟨*integer*⟩) | UINT(⟨*integer*⟩) | FLOAT(⟨*float*⟩) | REAL(⟨*float*⟩) | ⟨*constructValue*⟩ | ⟨*constructArrValue*⟩

⟨*bool*⟩ ::= true | false

⟨*constructValue*⟩ ::= construct(⟨*fieldValue*⟩ [, ⟨*fieldValue*⟩]*)

⟨*fieldValBlk*⟩ ::= label_const(⟨*ident*⟩, ⟨*value*⟩)

⟨*constructArrValue*⟩ ::= construct_array(⟨*value*⟩ [, ⟨*value*⟩]*)

5 Node Block

⟨*nodeBlk*⟩ ::= node(⟨*kind*⟩, ⟨*guid*⟩, ⟨*ident*⟩, ⟨*comment*⟩, ⟨*paramBlk*⟩, ⟨*returnBlk*⟩, ⟨*bodyBlk*⟩)

⟨*kind*⟩ ::= node | function

⟨*guid*⟩ ::= [0-9a-z-]*

⟨*paramBlk*⟩ ::= params([⟨*varDeclsStmt*⟩ [, ⟨*varDeclsStmt*⟩]*])

⟨*returnBlk*⟩ ::= returns([⟨*varDeclsStmt*⟩ [, ⟨*varDeclsStmt*⟩]*])

⟨*varDeclsStmt*⟩ ::= var_decls(vars(⟨*ident*⟩ [, ⟨*ident*⟩]*), ⟨*type*⟩, ⟨*comment*⟩)

⟨*bodyBlk*⟩ ::= body([⟨*localVarsBlk*⟩,] ⟨*assignBlk*⟩)

⟨*localVarsBlk*⟩ ::= localvars(⟨*varDeclsStmt*⟩ [,⟨*varDeclsStmt*⟩]*)

⟨*assignBlk*⟩ ::= [⟨*assignStmt*⟩ [,⟨*assignStmt*⟩]*]

⟨*assignStmt*⟩ ::= =(lvalue(⟨*lhs*⟩ [, ⟨*lhs*⟩]*), ⟨*expr*⟩, ⟨*guidop*⟩, ⟨*guidVal*⟩, ⟨*imported*⟩, ⟨*importCode*⟩)

⟨*lhs*⟩ ::= anonymous_id | ID(⟨*ident*⟩, ⟨*type*⟩, ⟨*clock*⟩)

⟨*guidop*⟩ ::= ⟨*ident*⟩ | NOCALL

⟨*guidVal*⟩ ::= ⟨*guid*⟩ | NOGUID

⟨*imported*⟩ ::= NOIMPORT | IMPORTED

⟨*importCode*⟩ ::= ⟨*integer*⟩

⟨*expr*⟩ ::= ⟨*atomExpr*⟩ | ⟨*binopExpr*⟩ | ⟨*unopExpr*⟩ | ⟨*ifExpr*⟩ | ⟨*switchExpr*⟩ | ⟨*tempoPreExpr*⟩ | ⟨*tempoArrowExpr*⟩ | ⟨*tempoFbyExpr*⟩ | ⟨*fieldAccessExpr*⟩ | ⟨*constructExpr*⟩ | ⟨*constructArrExpr*⟩ | ⟨*mixedConstructorExpr*⟩ | ⟨*arrDimExpr*⟩ | ⟨*arrIdxExpr*⟩ | ⟨*arrSliceExpr*⟩ | ⟨*listExpr*⟩ | ⟨*applyExpr*⟩ | ⟨*dynamicProjExpr*⟩

⟨*atomExpr*⟩ ::= ID(⟨*ident*⟩, ⟨*type*⟩, ⟨*clock*⟩) | ID(⟨*ident*⟩) | BOOL(⟨*bool*⟩) | CHAR(⟨*integer*⟩) | SHORT(⟨*integer*⟩) | USHORT(⟨*integer*⟩) | INT(⟨*integer*⟩) | UINT(⟨*integer*⟩) | FLOAT(⟨*float*⟩) | REAL(⟨*float*⟩) |

⟨*binopExpr*⟩ ::= ⟨*binop*⟩(⟨*type*⟩, ⟨*clock*⟩, ⟨*expr*⟩, ⟨*expr*⟩)

⟨*binop*⟩ ::= binop_add | binop_subtract | binop_multiply | binop_divide | binop_-div | binop_mod | binop_and | binop_or | binop_xor | binop_gt | binop_lt | binop_ge | binop_le | binop_eq | binop_neq

⟨*unopExpr*⟩ ::= ⟨*unop*⟩(⟨*type*⟩, ⟨*clock*⟩, ⟨*expr*⟩)

⟨*unop*⟩ ::= unop_shortcast | unop_intcast | unop_floatcast | unop_realcast | unop_-not | unop_pos | unop_neg

⟨*ifExpr*⟩ ::= if_expr(⟨*type*⟩, ⟨*clock*⟩, ⟨*expr*⟩, ⟨*expr*⟩, ⟨*expr*⟩)

⟨*switchExpr*⟩ ::= switch_expr(⟨*type*⟩, ⟨*clock*⟩, ⟨*expr*⟩, ⟨*caseStmt*⟩ [,⟨*caseStmt*⟩]*)

⟨*caseStmt*⟩ ::= case(⟨*value*⟩, ⟨*expr*⟩) | case(pattern_any, ⟨*expr*⟩)

⟨*tempoPreExpr*⟩ ::= tempo_pre(⟨*type*⟩, ⟨*clock*⟩, ⟨*expr*⟩)

⟨*tempoArrowExpr*⟩ ::= tempo_arrow(⟨*type*⟩, ⟨*clock*⟩, ⟨*expr*⟩, ⟨*expr*⟩)

⟨*tempoFbyExpr*⟩ ::= tempo_fby(⟨*type*⟩, ⟨*clock*⟩, ⟨*listExpr*⟩, ⟨*expr*⟩, ⟨*listExpr*⟩)

⟨*constructExpr*⟩ ::= construct(⟨*type*⟩, ⟨*clock*⟩, [⟨*labelExpr*⟩ ,]+)

⟨*labelExpr*⟩ ::= label_expr(⟨*ident*⟩, ⟨*expr*⟩)

⟨*constructArrExpr*⟩ ::= construct_array(⟨*type*⟩, ⟨*clock*⟩, ⟨*listExpr*⟩)

⟨*mixedConstructorExpr*⟩ ::= mixed_constructor(⟨*type*⟩, ⟨*clock*⟩, ⟨*expr*⟩, ⟨*labelIdxList*⟩, ⟨*expr*⟩)

⟨*labelIdxList*⟩ ::= (⟨*labelIdx*⟩[, lableIdx]*)

⟨*labelIdx*⟩ ::= struct_item(⟨*ident*⟩) | ⟨*expr*⟩

⟨*fieldAccessExpr*⟩ ::= field_access(⟨*type*⟩, ⟨*clock*⟩, ⟨*expr*⟩, ⟨*ident*⟩)

⟨*arrDimExpr*⟩ ::= array_dim(⟨*type*⟩, ⟨*clock*⟩, ⟨*expr*⟩, ⟨*intValue*⟩)

⟨*arrIdxExpr*⟩ ::= array_index(⟨*type*⟩, ⟨*clock*⟩, ⟨*expr*⟩, ⟨*intValue*⟩)

⟨*arrSliceExpr*⟩ ::= array_slice(⟨*type*⟩, ⟨*clock*⟩, ⟨*expr*⟩, ⟨*expr*⟩, ⟨*expr*⟩)

⟨*listExpr*⟩ ::= list_expr(⟨*expr*⟩ [, ⟨*expr*⟩]*) | list_expr()

⟨*dynamicProjExpr*⟩ ::= dynamic_project(⟨*type*⟩, ⟨*clock*⟩, ⟨*expr*⟩, (⟨*expr*⟩ [,⟨*expr*⟩]*), ⟨*expr*⟩)

⟨*applyExpr*⟩ ::= apply_expr(⟨*type*⟩, ⟨*clock*⟩, ⟨*applyBlk*⟩, ⟨*listExpr*⟩)

⟨*applyBlk*⟩ ::= ⟨*makeStmt*⟩ | ⟨*flattenStmt*⟩ | ⟨*highOrderStmt*⟩ | ⟨*prefixStmt*⟩ | ⟨*mapwDefaultStmt*⟩ | ⟨*mapwiDefaultStmt*⟩ | ⟨*foldwIfStmt*⟩ | ⟨*foldwiStmt*⟩

⟨*makeStmt*⟩ ::= make(⟨*ident*⟩, ⟨*type*⟩)

⟨*flattenStmt*⟩ ::= flatten(⟨*ident*⟩, ⟨*type*⟩)

⟨*highOrderStmt*⟩ ::= high_order(⟨*highOrderOp*⟩, ⟨*prefixStmt*⟩, ⟨*intValue*⟩)

⟨*prefixStmt*⟩ ::= prefix(⟨*ident*⟩, param_types(⟨*type*⟩[, ⟨*type*⟩]*), ret_types(⟨*type*⟩ [, ⟨*type*⟩]*)) | prefix(⟨*prefixBinOp*⟩) | prefix(⟨*prefixUnOp*⟩)

⟨*highOrderOp*⟩ ::= highorder_map | highorder_fold | highorder_mapfold | highorder_-mapi | highorder_foldi

⟨*prefixUnOp*⟩ ::= short$ | int$ | float$ | real$ | not$ | +$ | -$

⟨*prefixBinOp*⟩ ::= $+$ | $-$ | $*$ | $/$ | div | mod | and | or | xor | $=$ | $⟨⟩$ | $>$ | $>=$ | $<$ | $<=$

⟨*mapwDefaultStmt*⟩ ::= mapw_default(⟨*prefix*⟩, ⟨*intValue*⟩, ⟨*expr*⟩, ⟨*expr*⟩,)

⟨*mapwiDefaultStmt*⟩ ::= mapwi_default(⟨*prefix*⟩, ⟨*intValue*⟩, ⟨*expr*⟩, ⟨*expr*⟩,)

⟨*foldwIfStmt*⟩ ::= foldw_if(⟨*prefix*⟩, ⟨*intValue*⟩, ⟨*expr*⟩)

⟨*foldwiStmt*⟩ ::= foldwi(⟨*prefix*⟩, ⟨*intValue*⟩, ⟨*expr*⟩)