Language Grammar

⟨*Model*⟩ ::= ⟨*ProcClass*⟩ | ⟨*ProcDSL*⟩ | ⟨*SchDSL*⟩

⟨*ProcClass*⟩ ::= '`process`' ⟨*ID*⟩ ['`refines`' ⟨*ID*⟩] '`{`' [⟨*DefAttr*⟩] ⟨*DefBehavior*⟩ '`}`' '`configuration`' '`{`' [⟨*ProcessConfig*⟩] ⟨*ProcessInit*⟩ '`}`'

⟨*DefAttr*⟩ ::= '`attribute`' '`{`' (⟨*AttDef*⟩)* [⟨*Constraints*⟩] '`}`'

⟨*AttDef*⟩ ::= ⟨*ID*⟩ '`:`' '`type`' '`=`' ⟨*Type*⟩ ['`,`' '`value`' '`=`' ⟨*ListDef*⟩] '`,`' '`default`' '`=`' ⟨*Value*⟩ '`;`'

⟨*ListDef*⟩ ::= '`[`' ⟨*List*⟩ ('`,`' ⟨*List*⟩) * '`]`'

⟨*List*⟩ ::= ⟨*Range*⟩ | ⟨*BOOL*⟩ | ⟨*ID*⟩

⟨*Range*⟩ ::= ⟨*INT*⟩ '`..`' ⟨*INT*⟩

⟨*Value*⟩ ::= ⟨*BOOL*⟩ | ⟨*INT*⟩

⟨*DefBehavior*⟩ ::= ⟨*ProcType*⟩ | ⟨*ProcBehav*⟩

⟨*ProcType*⟩ ::= (⟨*ProcessType*⟩)*

⟨*ProcessType*⟩ ::= '`proctype`' ⟨*ID*⟩ '`{`' [⟨*Constraints*⟩] (⟨*ProcBehav*⟩)* '`}`'

⟨*ProcBehav*⟩ ::= '`behavior`' '`{`' (⟨*PBehav*⟩)* '`}`'

⟨*PBehav*⟩ ::= ⟨*Constructor*⟩ | ⟨*Method*⟩

⟨*Constructor*⟩ ::= '`constructor`' '`:`' ⟨*ID*⟩ '`(`' [⟨*PramList*⟩]'`)`' '`;`'

⟨*Method*⟩ ::= '`method`' '`:`' ⟨*ID*⟩ (('`(`' '`)`' '`;`') | ('`(`' ⟨*PramList*⟩ '`)`' '`{`' (⟨*AssignPara*⟩)* [⟨*Constraints*⟩] '`}`'))

⟨*AssignPara*⟩ ::= ⟨*ID*⟩ '`:`' '`value`' '`=`' ⟨*ListDef*⟩ '`;`'

⟨*Constraints*⟩ ::= '`constraint`' '`{`' (⟨*Constr*⟩) * '`}`'

⟨*Constr*⟩ ::= ⟨*Or*⟩ '`;`'

⟨*ProcDSL*⟩ ::= '`def`' '`process`' '`{`' [⟨*ProcAttr*⟩] ⟨*Process*⟩* '`}`' [⟨*ProcConf*⟩] [⟨*ProcInit*⟩]

⟨*ProcAttr*⟩ ::= '`attribute`' '`{`' ⟨*PAttr*⟩* '`}`'

⟨*PAttr*⟩ ::= ['`var`'|'`val`'] ⟨*Type*⟩⟨*ID*⟩ ('`,`'⟨*ID*⟩)* ['`=`'⟨*Value*⟩]'`;`'

⟨*Type*⟩ ::= '`int`' | '`byte`' | '`clock`'

⟨*Process*⟩ ::= ‘`proctype`’ ⟨*ID*⟩ ‘`(`’[⟨*PramList*⟩]‘`)`’ ‘`{`’ ⟨*AttAss*⟩* ‘`}`’

⟨*PramList*⟩ ::= ⟨*PramAss*⟩ (‘`;`’ ⟨*PramAss*⟩)*

⟨*PramAss*⟩ ::= ⟨*Type*⟩ ⟨*ID*⟩ (‘`,`’ ⟨*ID*⟩)* ‘`=`’ ⟨*Value*⟩

⟨*AttAss*⟩ ::= [‘`this`’ ‘`.`’] ⟨*ID*⟩ ‘`=`’ (⟨*Value*⟩ | ⟨*ID*⟩) ‘`;`’

⟨*ProcConf*⟩ ::= ‘`config`’ ‘`{`’ ⟨*PConf*⟩* ‘`}`’

⟨*PConf*⟩ ::= ⟨*SporadicP*⟩ | ⟨*PeriodicP*⟩

⟨*SporadicP*⟩ ::= ‘`sporadic`’ ‘`process`’ ⟨*Proc*⟩ ‘`in`’ ‘`(`’ ⟨*INT*⟩ ‘`,`’ ⟨*INT*⟩ ‘`)`’ [‘`limited`’ ⟨*INT*⟩] ‘`;`’

⟨*PeriodicP*⟩ ::= ‘`periodic`’ ‘`process`’ ⟨*Proc*⟩ ‘`offset`’ ‘`=`’ ⟨*INT*⟩ ‘`period`’ ‘`=`’ ⟨*INT*⟩ [‘`limited`’ ⟨*INT*⟩] ‘`;`’

⟨*Proc*⟩ ::= ⟨*ID*⟩ ‘`(`’ [⟨*Value*⟩ (‘`,`’ ⟨*Value*⟩)*] ‘`)`’

⟨*ProcInit*⟩ ::= ‘`init`’ ‘`{`’ ‘`[`’ ⟨*PSet*⟩ (‘`,`’ ⟨*PSet*⟩)* ‘`]`’ ‘`}`’‘`;`’

⟨*PSet*⟩ ::= ‘`{`’ ⟨*Proc*⟩ (‘`,`’ ⟨*Proc*⟩)* ‘`}`’

⟨*SchDSL*⟩ ::= ⟨*SchDef*⟩ [⟨*OrdDef*⟩] [⟨*Verify*⟩]

⟨*SchDef*⟩ ::= ‘`scheduler`’ ⟨*ID*⟩ ‘`(`’ [⟨*ParamList*⟩] ‘`)`’ [‘`refines`’ ⟨*ID*⟩] ‘`{`’ [⟨*Generate*⟩] [⟨*VarDef*⟩] [⟨*DatDef*⟩] [⟨*HandlerDef*⟩] [⟨*InterDef*⟩] ‘`}`’

⟨*Generate*⟩ ::= ‘`generate`’ ‘`{`’ ⟨*GenConfig*⟩ ⟨*GenComp*⟩‘`}`’

⟨*GenConfig*⟩ ::= ‘`configuration`’‘`{`’ [⟨*GenOption*⟩ ‘`;`’] [⟨*Dir*⟩ ‘`;`’] [⟨*FName*⟩ ‘`;`’] [⟨*FExt*⟩ ‘`;`’] ‘`test`’ (‘`program`’ | ‘`case`’ | ‘`data`’) ‘`=`’ ⟨*TestPart*⟩ ‘`}`’

⟨*GenOption*⟩ ::= ‘`option`’ ‘`=`’ ‘`{`’ ⟨*GenOpt*⟩ (‘`,`’ ⟨*GenOpt*⟩)* ‘`}`’

⟨*GenOpt*⟩ ::= ‘`Searching`’ | ‘`Error`’ | ‘`Property`’ | ‘`All`’

⟨*Dir*⟩ ::= ‘`directory`’ ‘`=`’ ⟨*STRING*⟩ ‘`;`’

⟨*FName*⟩ ::= ‘`file`’ ‘`name`’ ‘`=`’ ⟨*STRING*⟩ ‘`;`’

⟨*FExt*⟩ ::= ‘`file`’ ‘`extension`’ ‘`=`’ ⟨*STRING*⟩ ‘`;`’

⟨*TestPart*⟩ ::= ⟨*GenPart*⟩ (‘`+`’ ⟨*GenPart*⟩)*

⟨*GenPart*⟩ ::= ‘`(`’ [⟨*STRING*⟩ ‘`+`’] (⟨*ID*⟩ | ‘`init`’ | ‘`processes`’ | ‘`behaviors`’ | ‘`error`’) [‘`+`’ ⟨*STRING*⟩] ‘`)`’

⟨*GenComp*⟩ ::= ‘`component`’ ‘`{`’ (⟨*Comp*⟩)* [⟨*InitGen*⟩] [⟨*ProcGen*⟩] ‘`}`’;

⟨*Comp*⟩ ::= ⟨*ID*⟩ ‘`{`’ (⟨*Gen*⟩ |⟨*GenLn*⟩)* ‘`}`’

⟨*InitGen*⟩ ::= ‘`init`’ ‘`{`’ ⟨*Template*⟩ ‘`}`’

⟨*ProcGen*⟩ ::= ‘`process`’ ‘`{`’ ⟨*Template*⟩ ‘`}`’

⟨*Template*⟩ ::= [⟨*SetTemplate*⟩] ⟨*Behavior*⟩

⟨*SetTemplate*⟩ ::= '`template`' '`=`' ⟨*Expr*⟩ '`;`'

⟨*Behavior*⟩ ::= '`behavior`' '`=`' ⟨*EventTemp*⟩ ('`+`' ⟨*EventTemp*⟩)* '`;`'

⟨*EventTemp*⟩ ::= '`(`' [⟨*Expr*⟩ '`+`'] ⟨*Event*⟩ ['`+`' ⟨*Expr*⟩] '`)`'

⟨*VarDef*⟩ ::= '`variable`' '`{`' ⟨*VDec*⟩* '`}`'

⟨*VDec*⟩ ::= [⟨*IfDef*⟩] (⟨*VBlockDef*⟩ | ⟨*VOneDef*⟩)

⟨*IfDef*⟩ ::= '`#`' '`ifdef`' '`(`' ⟨*Expr*⟩ '`)`'

⟨*VBlockDef*⟩ ::= '`{`' ⟨*VOneDef*⟩* '`}`'

⟨*VOneDef*⟩ ::= ⟨*Type*⟩ ⟨*ID*⟩ ('`,`' ⟨*ID*⟩)* ['`=`' ⟨*Value*⟩] '`;`'

⟨*DatDef*⟩ ::= '`data`' '`{`' ⟨*DDef*⟩* '`}`'

⟨*DDef*⟩ ::= [⟨*IfDef*⟩] '`data`' (⟨*DBlockDef*⟩ | ⟨*DOneDef*⟩)

⟨*DBlockDef*⟩ ::= '`{`' ⟨*DOneDef*⟩* '`}`'

⟨*DOneDef*⟩ ::= ⟨*VOneDef*⟩ | ⟨*ColDef*⟩

⟨*ColDef*⟩ ::= ['`refines`'] '`collection`' ⟨*ID*⟩ ['`using`' ⟨*ID*⟩ ('`,`' ⟨*ID*⟩)*] ['`with`' ⟨*OrdType*⟩] '`;`'

⟨*OrdType*⟩ ::= '`lifo`' | '`fifo`'

⟨*HandlerDef*⟩ ::= '`event`' '`handler`' '`{`' ⟨*EventDef*⟩* '`}`'

⟨*EventDef*⟩ ::= ⟨*Event*⟩ '`(`' [⟨*ID*⟩] '`)`' '`{`' ⟨*IfDefStm*⟩* '`}`'

⟨*IfDefStm*⟩ ::= [⟨*IfDef*⟩] ⟨*Stm*⟩

⟨*Event*⟩ ::= '`select_process`' | '`new_process`' | '`clock`' | '`pre_take`' | '`post_take`' | '`action`'

⟨*InterDef*⟩ ::= '`interface`' '`{`' ⟨*InterFunc*⟩* '`}`'

⟨*InterFunc*⟩ ::= '`function`' ⟨*ID*⟩ '`(`' [⟨*IParamList*⟩] '`)`' '`{`' ⟨*Stm*⟩* '`}`'

⟨*IParamList*⟩ ::= ⟨*IParamDec*⟩ ('`,`' ⟨*IParamDec*⟩)*

⟨*IParamDec*⟩ ::= ⟨*Type*⟩ ⟨*ID*⟩

⟨*OrdDef*⟩ ::= '`comparator`' '`{`' [⟨*CVarDef*⟩] ⟨*CompDef*⟩* '`}`'

⟨*CVarDef*⟩ ::= '`variable`' '`{`' ⟨*VOneDef*⟩* '`}`'

⟨*CompDef*⟩ ::= '`comparetype`' ⟨*ID*⟩ '`(`' '`process`' ⟨*ID*⟩ '`,`' ⟨*ID*⟩ '`)`' '`{`' ⟨*Stm*⟩* '`}`'

⟨*Stm*⟩ ::= ⟨*SetTime*⟩ | ⟨*SetCol*⟩ | ⟨*Change*⟩ | ⟨*Move*⟩ | ⟨*Remove*⟩ | ⟨*Get*⟩ | ⟨*New*⟩ | ⟨*If*⟩ | ⟨*Loop*⟩ | ⟨*Block*⟩ | ⟨*Assert*⟩ | ⟨*Print*⟩ | ⟨*Return*⟩ | ⟨*Gen*⟩ |⟨*GenLn*⟩

⟨*SetTime*⟩ ::= '`time_slice`' '`=`' ⟨*Expr*⟩ '`;`'

⟨*SetCol*⟩ ::= '`return_set`' '`=`' ⟨*ID*⟩ '`;`'

⟨*Change*⟩ ::= ⟨*ChgUnOp*⟩ | ⟨*ChgExpr*⟩

⟨*ChgUnOp*⟩ ::= ⟨*QualName*⟩ ('`++`' | '`{{`') '`;`'

⟨*ChgExpr*⟩ ::= ⟨*QualName*⟩ '`=`' ⟨*Expr*⟩ '`;`'

⟨*QualName*⟩ ::= ⟨*ID*⟩ ['`.`'⟨*ID*⟩]

⟨*Move*⟩ ::= '`move`' ⟨*ID*⟩ to ⟨*ID*⟩ '`;`'

⟨*Remove*⟩ ::= '`remove`' ⟨*ID*⟩ '`;`'

⟨*Get*⟩ ::= '`get`' '`process`' '`from`' ⟨*ID*⟩ '`to`' '`run`' '`;`'

⟨*New*⟩ ::= '`new`' ⟨*Proc*⟩ ['`,`' ⟨*INT*⟩] '`;`'

⟨*If*⟩ ::= '`if`' '`(`' ⟨*Expr*⟩ '`)`' ⟨*Stm*⟩ ['`else`' ⟨*Stm*⟩]

⟨*Loop*⟩ ::= '`for`' '`each`' '`process`' ⟨*ID*⟩ '`in`' ⟨*ID*⟩ ⟨*Stm*⟩

⟨*Block*⟩ ::= '`{`' ⟨*Stm*⟩* '`}`'

⟨*Assert*⟩ ::= '`assert`' ⟨*Expr*⟩ '`;`'

⟨*Print*⟩ ::= '`print`' ⟨*Expr*⟩ '`;`'

⟨*Return*⟩ ::= '`return`' ⟨*OrderType*⟩ '`;`'

⟨*OrderType*⟩ ::= '`greater`' | '`less`' | '`equal`'

⟨*Gen*⟩ ::= '`gen`' [⟨*ID*⟩ '`,`'] ⟨*Expr*⟩ '`;`'

⟨*GenLn*⟩ ::= '`genln`' [⟨*ID*⟩ '`,`'] ⟨*Expr*⟩ '`;`'

⟨*Expr*⟩ ::= ⟨*Or*⟩

⟨*Or*⟩ ::= ⟨*And*⟩ ('`||`' ⟨*And*⟩)*

⟨*And*⟩ ::= ⟨*Equality*⟩ ('`&&`' ⟨*Equality*⟩)*

⟨*Equality*⟩ ::= ⟨*Equality*⟩ ('`==`' | '`!=`') ⟨*Compar*⟩

⟨*Compar*⟩ ::= ⟨*PlusMinus*⟩ ('`>=`' | '`<=`' | '`>`' | '`<`') ⟨*PlusMinus*⟩

⟨*PlusMinus*⟩ ::= ⟨*MulOrDiv*⟩ ('`+`' | '`-`') ⟨*MulOrDiv*⟩

⟨*MulOrDiv*⟩ ::= ⟨*MulOrDiv*⟩ ('`*`' | '`/`') ⟨*Primary*⟩

⟨*Primary*⟩ ::= '`(`'⟨*Expr*⟩'`)`' | '`!`'⟨*Primary*⟩ | ⟨*Empty*⟩ | ⟨*Null*⟩ | ⟨*InCol*⟩ | ⟨*Exist*⟩ | ⟨*GetID*⟩ | ⟨*HasName*⟩ | ⟨*Atomic*⟩

⟨*Empty*⟩ ::= ⟨*ID*⟩'`.`' '`isEmpty`' '`(`' '`)`'

⟨*Null*⟩ ::= ⟨*ID*⟩'`.`' '`isNull`' '`(`' '`)`'

⟨*InCol*⟩ ::= ⟨*ID*⟩'`.`' '`containsProcess`' '`(`' ⟨*STRING*⟩ '`)`'

⟨*Exist*⟩ ::= '`exists`' '`(`' ⟨*STRING*⟩ '`)`'

⟨*GetID*⟩ ::= '`get_pid`' '`(`' ⟨*STRING*⟩ '`)`'

⟨*HasName*⟩ ::= ⟨*ID*⟩ '`.`' hasName' '`(`' ⟨*STRING*⟩ '`)`'

⟨*Atomic*⟩ ::= ⟨*Value*⟩ | ⟨*QualName*⟩ | ⟨*SysVar*⟩

⟨*SysVar*⟩ ::= '`Sys`' '`(`' ⟨*ID*⟩ '`)`'

⟨*Verify*⟩ ::= '`verify`' '`{`' [⟨*CTL_AT*⟩] ⟨*RTCTL*⟩ '`}`'

⟨*CTL_AT*⟩ ::= '`@`' ⟨*Expr*⟩ '`:`'

⟨*RTCTL*⟩ ::= '`(`' ⟨*Expr*⟩ '`)`' | '`not`' ⟨*RTCTL*⟩ | '`or`' ⟨*RTCTL*⟩ ⟨*RTCTL*⟩ | '`implies`' ⟨*RTCTL*⟩ ⟨*RTCTL*⟩ | '`AX`' ⟨*RTCTL*⟩ | '`AF`' [⟨*LTE*⟩] ⟨*RTCTL*⟩ | '`AG`' [⟨*LTE*⟩] ⟨*RTCTL*⟩ | '`EX`' ⟨*RTCTL*⟩ | '`EF`' [⟨*LTE*⟩] ⟨*RTCTL*⟩ | '`EG`' [⟨*LTE*⟩] ⟨*RTCTL*⟩ | '`AU`' [⟨*LTE*⟩] ⟨*RTCTL*⟩ ⟨*RTCTL*⟩ | '`EU`' [⟨*LTE*⟩] ⟨*RTCTL*⟩ ⟨*RTCTL*⟩

⟨*LTE*⟩ ::= '`<=`' ⟨*INT*⟩

- We note that some terms, such as `<ID>`, `<STRING>`, `<INT>`, `<BOOL>`, are not shown in the grammar.
- The '`val`' ('`var`') keyword for defining an attribute of the process indicates that the value of this attribute is unchangeable (changeable). Only the values assigning to the changeable attributes are stored in the system state.
- The `<IfDef>` statement is used for initializing the scheduler based on the condition `<Expr>`. This statement allows us to deal with parameterizing the scheduling policy.
- We also support reusing the specification by introducing '`refines`' keyword. If scheduler `B` '`refines`' scheduler `A`, all of the data structures and the event handlers of `A` are inherited by `B`; however, `B` can redefine them, add more data structures and handle its new events. It is similar to the inheritance in object-oriented programming. With a collection, '`refines`' means redefining its ordering method.