```
grammar Stipula ;
@header {
    package parser;
}
@lexer::members {
   //there is a much better way to do this, check the ANTLR guide
   //I will leave it like this for now just becasue it is quick
   //but it doesn't work well
   public int lexicalErrors=0;
}

/*------------------------------------------------------------------
 * PARSER RULES
 *------------------------------------------------------------------*/

prog :  STIPULA contract_id = ID CLPAR (assetdecl)? (fielddecl)? agreement (fun)+ CRPAR ;

agreement : (AGREEMENT LPAR party (COMMA party)* RPAR LPAR vardec (COMMA vardec)* RPAR CLPAR
(assign)+ CRPAR IMPL AT state);

assetdecl :  ASSET idAsset+=ID (COMMA idAsset+=ID)* ;

fielddecl : FIELD idField+=ID (COMMA idField+=ID)* ;

fun :  ((AT state)* (party (COMMA party)* | TILDE) COLON funId=ID LPAR (vardec ( COMMA vardec)* )?
RPAR SLPAR (assetdec ( COMMA assetdec)* )? SRPAR (LPAR prec RPAR)? CLPAR (stat)* (event)* CRPAR
IMPL AT state )   ;

assign : (party (COMMA party)* COLON vardec (COMMA vardec)*);

dec : (ASSET | FIELD) ID  ;

type :  INTEGER | DOUBLE | BOOLEAN | STRING ;

state : ID;

party : ID;

vardec  : ID ;

assetdec  : ID ;

varasm     : vardec ASM expr ;

stat    :   left=value operator=ASSETUP right=ID (COMMA rightPlus=ID)?
          | left=value operator=FIELDUP right=(ID | EMPTY)
          | ifelse
          ;

ifelse : (IF LPAR cond=expr RPAR CLPAR ifBranch+=stat (ifBranch+=stat)* CRPAR (ELSEIF
condElseIf+=expr CLPAR elseIfBranch+=stat (elseIfBranch+=stat)* CRPAR)* (ELSE CLPAR
elseBranch+=stat (elseBranch+=stat)* CRPAR )?);

event   :   expr TRIGGER AT ID CLPAR stat* CRPAR IMPL AT ID
          ;

prec  : expr
      ;
```

```
expr    :  ('-')? left=term (operator=(PLUS | MINUS | OR) right=expr)?
      ;

term   : left=factor (operator=(TIMES | DIV  | AND) right=term)?
      ;

factor : left=value (operator = (EQ | LE | GE | LEQ | GEQ | NEQ ) right=value)?
      ;

value  :  number
      | ID
      | NOW
      | LPAR expr RPAR
      | RAWSTRING
      | EMPTY
      | (TRUE | FALSE)
      ;

real : number DOT number ;

number : INT | REAL ;


/*------------------------------------------------------------------
 * LEXER RULES
 *------------------------------------------------------------------*/
SEMIC   : ';' ;
COLON   : ':' ;
COMMA   : ',' ;
EMPTY   : '_' ;
DOT     : '.' ;
EQ      : '==' ;
NEQ     : '!=' ;
IMPL    : '=>' ;
ASM     : '=' ;
ASSETUP :  '-o' ;
FIELDUP : '->' ;
PLUS    : '+' ;
MINUS   : '-' ;
TIMES   : '*' ;
DIV     : '/' ;
AT         : '@' ;
TILDE   : '~';
TRUE    : 'true' ;
FALSE   : 'false' ;
LPAR    : '(' ;
RPAR    : ')' ;
SLPAR   : '[' ;
SRPAR   : ']' ;
CLPAR   : '{' ;
CRPAR   : '}' ;
LEQ    : '<=';
GEQ    : '>=';
LE     : '<';
GE     : '>';
OR     : '||';
AND    : '&&';
```

```
NOT    : '!';
NOW     : 'now' ;
TRIGGER : '>>';
IF     : 'if' ;
ELSEIF   : 'else if' ;
ELSE   : 'else' ;
STIPULA : 'stipula';
ASSET : 'assets' ;
FIELD : 'fields' ;
AGREEMENT : 'agreement';
INTEGER : 'int' ;
DOUBLE : 'real' ;
BOOLEAN : 'bool' ;
STRING : 'string' ;
PARTY : 'party' ;
INIT : 'init' ;

RAWSTRING : '\'' ~('\'')+ '\'' | '"' ~('"')+ '"' ;

INT : '0' | [1-9] [0-9]* ;

REAL : [0-9]* '.' [0-9]+ ;

WS
 : [ \t\r\n] -> skip
 ;

//IDs
fragment CHAR  : 'a'..'z' |'A'..'Z' ;
ID : CHAR (CHAR | INT | EMPTY)* ;

OTHER
 : .
 ;

//ESCAPED SEQUENCES
LINECOMENTS    : '//' (~('\n'|'\r'))* -> skip;
BLOCKCOMENTS    : '/*'( ~('/'|'*')|'/'~'*'|'*'~'/'|BLOCKCOMENTS)* '*/' -> skip;

 //VERY SIMPLISTIC ERROR CHECK FOR THE LEXING PROCESS, THE OUTPUT GOES DIRECTLY TO THE TERMINAL
 //THIS IS WRONG!!!!
ERR     : . { System.out.println("Invalid char: "+ getText()); lexicalErrors++; } ->
channel(HIDDEN);
```