```
grammar HOStipula ;

@lexer::members {

 * PARSER RULES

prog :  STIPULA contract_id = ID CLPAR (assetdecl)? (fielddecl)? INIT init_state = ID 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 (',' idAsset+=ID)* ;

fielddecl : FIELD idField+=ID (',' idField+=ID)* ;

fun :  ((AT state)* party (COMMA party)* COLON funId=ID LPAR (vardec ( COMMA vardec)* )? RPAR SLPAR (assetdec
( COMMA assetdec)* )? SRPAR ( body | hobody) ) ;

body : (LPAR prec RPAR)? CLPAR (stat)+ SEMIC (events)+ CRPAR IMPL AT state ;

hobody  :  HOLPAR HID HORPAR   ;

hocode : ('parties' party (COMMA party)*)? (assetdecl)? (fielddecl)? fun* CLPAR (stat)+ SEMIC (events)+ CRPAR IMPL
AT state   ;

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

dec : (ASSET | FIELD) ID  ;

type :  INTEGER | DOUBLE | BOOLEAN | STRING ;

state : ID;
```

```
34
35 party : ID;
36
37 vardec : ID ;
38
39 assetdec : ID ;
40
41 varasm : vardec ASM expr ;
42
43 stat    :      EMPTY
44             |   left=value operator=ASSETUP right=ID (COMMA rightPlus=ID)?
45             |   left=value operator=FIELDUP right=(ID | EMPTY)
46             | ifelse
47
48             ;
49
50 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 )?);
51
52 events  :     EMPTY
53             | ( expr TRIGGER AT ID CLPAR stat+ CRPAR IMPL AT ID )
54             ;
55
56 prec  : expr
57       ;
58
59 expr    :  ('-')? left=term (operator=(PLUS | MINUS | OR) right=expr)?
60       ;
61
62 term   : left=factor (operator=(TIMES | DIV  | AND) right=term)?
63       ;
64
```

```
65 factor : left=value (operator = (EQ | LE | GE | LEQ | GEQ | NEQ ) right=value)?
66       ;
67
68 value  :  number
69       | ID
70       | NOW
71       | LPAR expr RPAR
72       | RAWSTRING
73       | EMPTY
74       | (TRUE | FALSE)
75       ;
76
77 real : number DOT number ;
78
79 number : INT | REAL ;
80
81
83  * LEXER RULES
85 SEMIC    : ';' ;
86 COLON    : ':' ;
87 COMMA    : ',' ;
88 DOT      : '.' ;
89 EQ       : '==' ;
90 NEQ      : '!=' ;
91 IMPL     : '==>' ;
92 ASM      : '=' ;
93 ASSETUP  :  '-o' ;
94 FIELDUP  : '->' ;
95 PLUS     : '+' ;
96 MINUS    : '-' ;
97 TIMES    : '*' ;
98 DIV      : '/' ;
```

```
 99 AT       : '@' ;
100 TRUE     : 'true' ;
101 FALSE    : 'false' ;
102 LPAR     : '(' ;
103 RPAR     : ')' ;
104 SLPAR    : '[' ;
105 SRPAR    : ']' ;
106 HOLPAR   : '([' ;
107 HORPAR   : '])' ;
108 CLPAR    : '{' ;
109 CRPAR    : '}' ;
110 LEQ      : '<=';
111 GEQ      : '>=';
112 LE       : '<';
113 GE       : '>';
114 OR       : '||';
115 AND      : '&&';
116 NOT      : '!';
117 EMPTY    : '_' ;
118 NOW      : 'now' ;
119 TRIGGER  : '>>';
120 IF      : 'if' ;
121 ELSEIF    : 'else if' ;
122 ELSE    : 'else' ;
123 STIPULA : 'stipula';
124 ASSET : 'asset' ;
125 FIELD : 'field' ;
126 AGREEMENT : 'agreement';
127 INTEGER : 'int' ;
128 DOUBLE : 'real' ;
129 BOOLEAN : 'bool' ;
130 STRING : 'string' ;
```

```
131 PARTY : 'party' ;
132 INIT : 'init' ;
133
134 RAWSTRING : '\'' ~('\'')+ '\'' | '"' ~('"')+ '"' ;
135
136 INT : '0' | [1-9] [0-9]* ;
137
138 REAL : [0-9]* '.' [0-9]+ ;
139
140 WS
141  : [ \t\r\n] -> skip
142  ;
143
144 //IDs
145 fragment CHAR  : 'a'..'z' |'A'..'Z' ;
146 ID : CHAR (CHAR | INT | EMPTY)* ;
147 HID: 'A'..'Z' ;
148
149 OTHER
150  : .
151  ;
152
153 //ESCAPED SEQUENCES
154 LINECOMENTS    : '//' (~('\n'|'\r'))* -> skip;
155 BLOCKCOMENTS    : '/*'( ~('/'|'*')|'/'~'*'|'*'~'/'|BLOCKCOMENTS)* '*/' -> skip;
156
157  //VERY SIMPLISTIC ERROR CHECK FOR THE LEXING PROCESS, THE OUTPUT GOES DIRECTLY TO THE TERMINAL
158  //THIS IS WRONG!!!!
159 ERR     : . { System.out.println("Invalid char: "+ getText()); lexicalErrors++; } -> channel(HIDDEN);
```