Week 5- AUT Assignment

Group 3

Table of contents

Problem description

Assignment 5: "system engineering"

- write a parser that converts the z3 output of nfa.txt into GraphViz input
  - note: although you don't need all output, still make a grammar with meaningful rules to parse the complete output
- run the GraphViz executable and show the picture
- tips:
  - first make a grammar for z3's output of sudokuA.txt (in week 3) and sudokuB.txt (in week 4)
  - we recommend to use a listener instead of a visitor
- references for z3:
  - https://rise4fun.com/Z3/tutorial/guide
  - http://smtlib.cs.uiowa.edu/papers/smt-lib-reference-v2.6-r2017-07-18.pdf (see: chapter 3 "Syntax" (in particular: paragraph 3.9.1))

Implementation

For this week we have been tasked to write a parser that converts z3 output into GraphViz input.

First, we decided to use listeners instead of visitors, as in this case it seemed easier to walk the whole tree than to visit different branches. This turned out to be a good case distinction to see when the listeners or visitors are useful depending on what we want to do.

Afterwards, we had to reimplement our Z3 grammar and make sure it is well structured. The implementation is provided alongside this document.

The GraphViz graph can be found in the Output section.

```
grammar MyGrammar;

// rules
myStart        : statement EOF;

statement : checkSatResponse PARANL checkModelResponse PARANR;

checkSatResponse    : SAT | UNSAT | UNKNOWN;

checkModelResponse  :   MODEL? declarFun* ;
```

```
declarFun : PARANL DEFINE_FUN ID PARANL parameter* PARANR types expr* PARANR
# StatementFunction;

parameter: PARANL ID types PARANR;

types: STRING | BOOL | INT;

expr:
       ID                  # ValueVariable
     | BOOLEAN             # ValueBoolean
     | NUMBER              # ValueBasicNumber
     | TEXT                # ValueString
     | comparisonExpr      # ValueComparisonExpresssion
     | iteExpr             # StatementIfElse
     | logicalExpr         # ValueLogicalExpr
     | letExpr             # ValueLetExpr
     | mathExpr            # ValueMathExpr
     | func_call           # ValueFunc_call
     ;

func_call: PARANL (ID.PLUS | ID) expr*? PARANR;


iteExpr: PARANL ITE expr expr expr PARANR;

letExpr: PARANL LET PARANL expr+ PARANR expr PARANR;

logicalExpr :  PARANL OR expr*  PARANR
            | PARANL AND expr* PARANR
            | PARANL NOT expr PARANR;

comparisonExpr:
             PARANL LESS_THAN expr expr PARANR
           | PARANL GREATER_THAN expr expr PARANR
           | PARANL GREATER_THAN_OR_EQUAL expr expr PARANR
           | PARANL LESS_THAN_OR_EQUAL expr expr PARANR
           | PARANL EQUAL expr expr PARANR
           ;

mathExpr : PARANL PLUS expr* PARANR
        | PARANL MINUS expr* PARANR
        | PARANL DIVIDE expr* PARANR
        | PARANL MULTIPLY expr* PARANR;
```

```
//tokens
MODEL                   : 'model';
UNKNOWN                 : 'unknown';
ITE                     : 'ite';
GREATER_THAN            : '>' ;
LESS_THAN               : '<' ;
GREATER_THAN_OR_EQUAL   : '>=' ;
LESS_THAN_OR_EQUAL      : '<=' ;
PLUS                    :  '+';
MINUS                   :  '-';
DIVIDE                  :  '/';
MULTIPLY                :  '*';
STRING                  : 'String';
BOOL                    : 'Bool';
AND                     : 'and';
OR                      :  'or';
NOT                     : 'not';
LET                     :  'let';
EQUAL                   :  '=';
INT                     : 'Int';
NUMBER                  : [0-9]+;
TEXT                    : '"' ~('\r' | '\n' | '"')* '"';
UNSAT                   : 'unsat';
SAT                     : 'sat';
DEFINE_FUN              : 'define-fun';
BOOLEAN                 : 'true'| 'false';
ID                      :  [_A-Za-z][A-Za-z_!0-9.]* ;
PARANL                  : '(';
PARANR                  : ')';
NEWLINE                 : [ \t\r\n]+ -> skip;
```

Output

Below the GraphViz and Parse Tree output can be seen.