A quick guide to Verifier for Integer Assignment Programs (VIAP)

What is VIAP?

VIAP translates a program to first-order logic with quantifiers on natural numbers following the method recently proposed by *Fangzhen Lin*. Once translated to a first-order theory, properties of the program can then be proved using induction (because of the quantifiers on natural numbers) and other methods.

System File

$VIAP$ is developed using $python(2.7.11)$ and is completely independent of any operating system.
$VIAP$ source code is available on *github*, following location
https://github.com/VerifierIntegerAssignment/VIAP

System Requirement

User needs to make sure that the following packages are installed in the system to execute $VIAP$.

- Python 2.7.11
  Can be download from https://www.python.org/downloads/release/python-2711/
- sympy -
  pip install sympy[1]
  For More Details-http://www.sympy.org/en/index.html
- pyparsing -
  pip install pyparsing
  For More Details-http://pyparsing.wikispaces.com/
- regex -
  pip install regex
  For More Details-http://pyparsing.wikispaces.com/
- plyj -
  pip install plyj
  For More Details-https://github.com/musiKk/plyj
- wolframalpha -
  pip install wolframalpha
  For More Details-https://pypi.python.org/pypi/wolframalpha

External Solvers

$VIAP$ completely relies on external (SMT) solvers to prove the properties of a program. The current version of $VIAP$ support only $z3$ SMT solver.
To install z3

- z3 binaries are available at https://github.com/Z3Prover/z3
- Install it following the instruction of $README.md$.
- Set the path of z3.py of $z3$ in the system.
  - Windows:
    $MyComputer$ > $Properties$ > $Advanced\ System\ Settings$ > $Environment\ Variables$ > under system variables create a new Variable called $PYTHONPATH$ if not present and add location of z3.py file present in the installation directory of z3. If $PYTHONPATH$ is already present as a system variable, then append the location.
  - Linux & Mac OS-X:
    To set path in *Linux* and $Mac\ OS-X$, user need execute following instruction
    export PYTHONPATH=$PYTHONPATH :location of z3.py

How to setup Environment in Windows,Linux & Mac OS-X

The sources of $VIAP$ can be download by cloning the $VIAP$ repository:

- git clone https://github.com/VerifierIntegerAssignment/VIAP.git
  Cloning into 'VIAP'...
- cd VIAP/sourceCode
- Set properties *timeout* and *app_id* to values appreciative values.
  - *timeout* : Time out period of z3.(in millisecond). Default value is 60000.
  - *app_id* : application ID of wolfram mathematica web services. If user don't set the value of *app_id*, then wolfram mathematica module will remain inactive.If User assign value $None$ to it, then wolfram mathematica web services will be disable in the system. If user don't have internet connectivity in the system where $VIAP$, the user must disable wolfram mathematica web services. Otherwise it will return error. Disabling wolfram mathematica web services reduce the capability of VIAP
- Open python interpreter by typing the command *python*
- Execute the *viap.py* file in interpreter by typing the command $execfile('viap.py')$

Run Testsuite

After execution of *viap.py*, user can run Test suit by using following command. But before that copy benchmark directory to the same directory of file *testsuit.py*
Execute the *testsuit.py* file in interpreter by typing the command $execfile('testsuit.py')$

List of Command

translate(filepath)

translate command translates a computer program, P, which is a given in the file path, to a set $A(P, \vec{X})$, of first order logic axioms using the translation algorithm given in [1].This command returns a plain $Python$ object to store information about axioms.

Example 0.1. *The program P to find sum of natural numbers Using while loop.*

```
public void NSeries1(int X) {

int sum,i;
sum=0;
i=0;
while(i<X)
{
i=i+1;
sum=sum+i;

}

}
```

After application of translation,translate(P), user will get the following equations.
Output in normal notation:
1. Frame axioms:

```
X1 = X
```

2. Output equations:

```
i1 = (_N1+0)
sum1 = (((((_N1**2)+((2*_N1)*0))+_N1)+(2*0))/2)
```

3. Other axioms:

```
(_N1>=(X-0))
(_n1<_N1) -> ((_n1+0)<X)
```

displayAxioms(axiom)

Display axioms stored in axiom.

displayInputVariables(axiom)

Display Input Variables Information stored in axiom.

[1]Install pip using the instruction from hbmhttps://pip.pypa.io/en/stable/installing/

prove(axiom,pre_condition,post_condition)

- *axiom* is the plain Python object to store information about axioms returned by *translate* command.
- *pre_condition* is the set of pre-condition.
- *post_condition* is the set of post-condition user want to prove.

Output of the command can be one of the following

- Successfully Proved .
- Failed to Prove .
- Display counter example SMT solver return .

Example 0.2.
- *axiom contains the set of translated axioms of program P of Example 0.1.*
- *pre_condition=['X >= 0']*
- *post_condition=['sum1==X*(X+1)/2"]*

$prove(axiom, pre_condition, post_condition)$ *system tried to prove post-conditions according to strategies described in the paper. Result of example is - Successfully Proved.*

prove1(axiom,pre_condition,post_condition,flag)

- *axiom* is the plain Python object to store information about axioms returned by *translate* command.
- *pre_condition* is the set of pre-condition.
- *post_condition* is the set of post-condition user want to prove.
- If flag=1, then system use strategy 1 described in the paper. If flag=2, then system use strategy 2(Induction over *_n*) described in the paper.

Output of the command can be one of the following

- Successfully Proved .
- Failed to Prove .
- Display counter example SMT solver return .