

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/>
- pycparser -
*pip install pycparser*¹
For More Details-<https://github.com/eliben/pycparser>
- sympy -
*pip install sympy*²
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

prove_auto(filepath)

prove_auto command translates a computer program, *P*, which is a given in the file path, to a set $A(P, \bar{X})$, of first order logic axioms using the translation algorithm given in [1] and using those axioms tried to prove the assertions.

Example 0.1. The program *bhmr2007_true – unreachable – call.i* is from *SV-Comp benchmarks*.

```
extern void __VERIFIER_error(void);
extern void __VERIFIER_assume(int);
void __VERIFIER_assert(int cond) {
    if (!(cond)) {
        ERROR: __VERIFIER_error();
    }
    return;
}
int __VERIFIER_nondet_int();
int main() {
    int x = 1;
    int y = 0;
    while (y < 1000 && __VERIFIER_nondet_int()) {
        x = x + y;
        y = y + 1;
    }
    __VERIFIER_assert(x >= y);
    return 0;
}
```

After application of translation, *prove_auto(P)*, user will get the following equations and results.

Output in normal notation: Output for main: Output in prefix notation: 1. Frame axioms:

X1 = X

2. Output equations:

²Install pip using the instruction from <https://pip.pypa.io/en/stable/installing/>

```

y1 = (_N1+0)
x1 = (((((_N1**2)+((2*_N1)*0))-_N1)+(2*1))/2)
main = 0

```

3. Other axioms:

```

((( _N1+0)>=1000) or ( __VERIFIER_nondet_int<=0))
(_n1<_N1) -> (((_n1+0)<1000) and ( __VERIFIER_nondet_int>0))-1_FAILED1(cond) = ite((cond<=0),1,0)
__VERIFIER_assert(cond) = 0

```

4. Assumption :

5. Assertion :

```

((((((_N1**2)+((2*_N1)*0))-_N1)+(2*1))/2)>=(_N1+0))

```

Output for `--VERIFIER_assert`: Output in prefix notation:

Output in normal notation: 1. Frame axioms:

```
cond1(cond) = cond
```

2. Output equations:

```

1_FAILED1(cond) = ite((cond<=0),1,0)
__VERIFIER_assert(cond) = 0

```

3. Other axioms:

4. Assumption :

5. Assertion :

----- *ProvingProcess* -----

FunctionName --main

AssertionToProve : $((-N1 + N1 * N1 + 2/2) \geq N1)$

SuccessfullyProved

Output of the command can be one of the following

- Successfully Proved .
- Unknown .
- Display counter example SMT solver return which is represented in as witness automata.