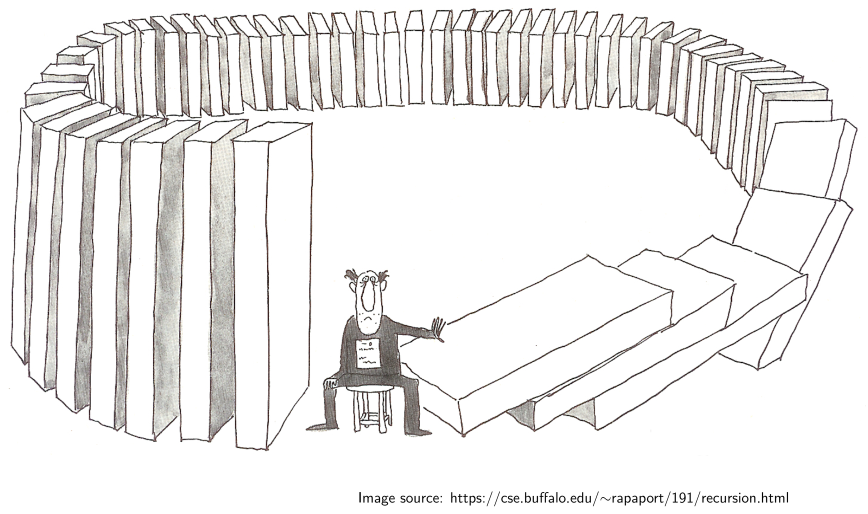


Verifying Array Programs with Full-program Induction

Supratik Chakraborty, Ashutosh Gupta, Divyesh Unadkat

Indian Institute of Technology Bombay, Mumbai, India

Email: {supratik,akg,divyesh}@cse.iitb.ac.in

Goal

Verify *(a class of)* quantified as well as quantifier-free post-conditions,

- programs manipulating **arrays** and **scalars** in loops
- parameterized by a special variable **N**
- quantified invariants over arrays or scalars **not** available
- use black-box back-ends such as **SMT** solver

Motivating Example

```
assume(true);
void Simple(int N) {
  int A[N], sum=0;
  for (int i=0; i<N; i++)
    A[i] = 1;
  for (int i=0; i<N; i++)
    sum = sum + A[i];
}
assert(sum==N);
```

Full-program Induction

Goal: Prove $\{\varphi_N\}$ P_N $\{\psi_N\}$ holds

Base case: Prove $\{\varphi_1\}$ P_1 $\{\psi_1\}$

Hypothesis:Assume $\{\varphi_{N-1}\}$ P_{N-1} $\{\psi_{N-1}\}$

Inductive step:
Compute difference pre-condition $\partial\varphi_N$ such that, $\varphi_N \rightarrow (\varphi_{N-1} \wedge \partial\varphi_N)$ and $\{\partial\varphi_N\}$ P_{N-1} $\{\partial\varphi_N\}$ hold

Compute $\partial\mathsf{P}_N$ such that, $\mathsf{P}_{N-1};\partial\mathsf{P}_N \equiv \mathsf{P}_N$ holds

Prove $\{\partial\varphi_N \wedge \psi_{N-1}\}$ $\partial\mathsf{P}_N$ $\{\psi_N\}$

Base Case: N=1

- Unrolled program *easy* for **SMT** solvers

Computing Difference Program

```
void Simple(int N) {
  int A_Nm1[N-1], A[N];
  int sum_Nm1=0, sum=0;
  for (int i=0; i<N-1; i++)
    A_Nm1[i] = 1;
  for (int i=0; i<N-1; i++)
    sum_Nm1 = sum_Nm1 + A_Nm1[i];

  for (int i=0; i<N-1; i++)
    A[i] = A_Nm1[i] + (1-1);
  A[N-1] = 1;
  sum = sum_Nm1;
  for (int i=0; i<N-1; i++)
   sum = sum + (A[i] - A_Nm1[i]);
  sum = sum + A[N-1];
}
```

Simplified Difference Program

```
assume(true);
void Simple(int N) {
  int A[N], sum=0;
  for (int i=0; i<N-1; i++)
    A[i] = 1;
  for (int i=0; i<N-1; i++)
    sum = sum + A[i];
}
assume(sum==N-1);
    A[N-1] = 1;
    sum = sum + A[N-1];
assert(sum==N);
```

Syntactic Restrictions

$$
\begin{aligned}
\mathsf{PB} &::= \mathsf{St} \mid \mathsf{St};\mathsf{St}\\
\mathsf{St} &::= v := \mathsf{E} \mid v_N := \mathsf{E}_N \mid A[\mathsf{E}_N] := \mathsf{E}_N \mid A[\mathsf{E}] := \mathsf{E} \mid\\
&\quad\ \mathbf{assume}(\mathsf{BoolE}) \mid \mathbf{if}(\mathsf{BoolE})\ \mathbf{then}\ \mathsf{St}\ \mathbf{else}\ \mathsf{St} \mid\\
&\quad\ \mathbf{for}\ (\ell := 0;\ \ell < N;\ \ell := \ell+1)\ \ \{\mathsf{St}\}\\
\mathsf{E}_N &::= \mathsf{E}_N\ \mathrm{op}\ \ \mathsf{E} \mid \mathsf{E}\ \mathrm{op}\ \ \mathsf{E}_N \mid \mathsf{E}_N\ \mathrm{op}\ \ \mathsf{E}_N \mid\\
&\quad\ A[\mathsf{E}_N] \mid v_N \mid N\\
\mathsf{E} &::= \mathsf{E}\ \mathrm{op}\ \ \mathsf{E} \mid A[\mathsf{E}] \mid v \mid \ell \mid \mathsf{c}\\
\mathsf{BoolE} &::= \mathsf{E}\ \mathrm{relop}\ \mathsf{E} \mid \mathsf{BoolE}\ \mathrm{AND}\ \mathsf{BoolE} \mid\\
&\quad\ \mathrm{NOT}\ \mathsf{BoolE} \mid \mathsf{BoolE}\ \mathrm{OR}\ \mathsf{BoolE}
\end{aligned}
$$

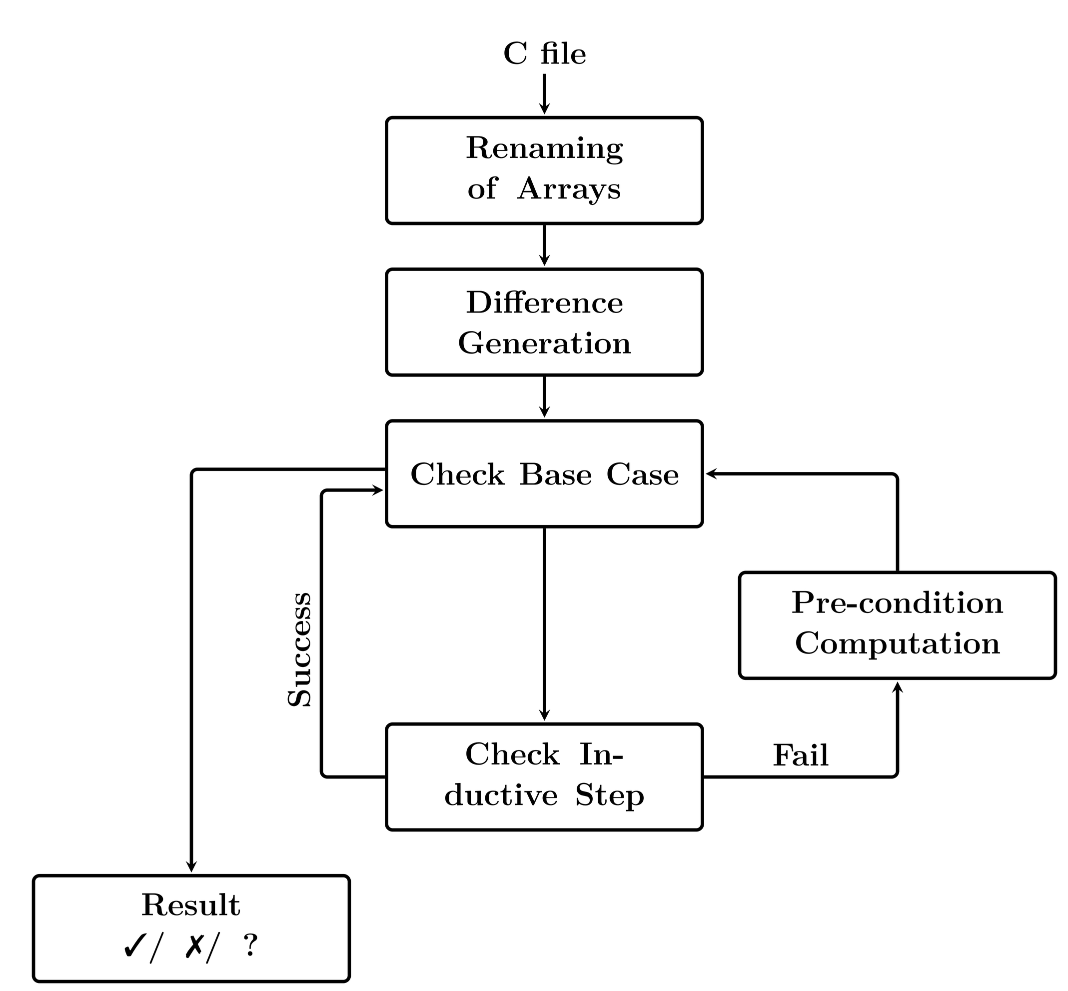
Need for Pre-conditions

```
assume(true);
void PolyCompute(int N) {
  int A[N], B[N], C[N];
  A[0]=6;  B[0]=1;  C[0]=0;
  for (int i=1; i<N; i++)
    A[i] = A[i-1] + 6;
  for (int i=1; i<N; i++)
    B[i] = B[i-1] + A[i-1];
  for (int i=1; i<N; i++)
    C[i] = C[i-1] + B[i-1];
}
assert(∀i∈[0,N),C[i]=i³);
```

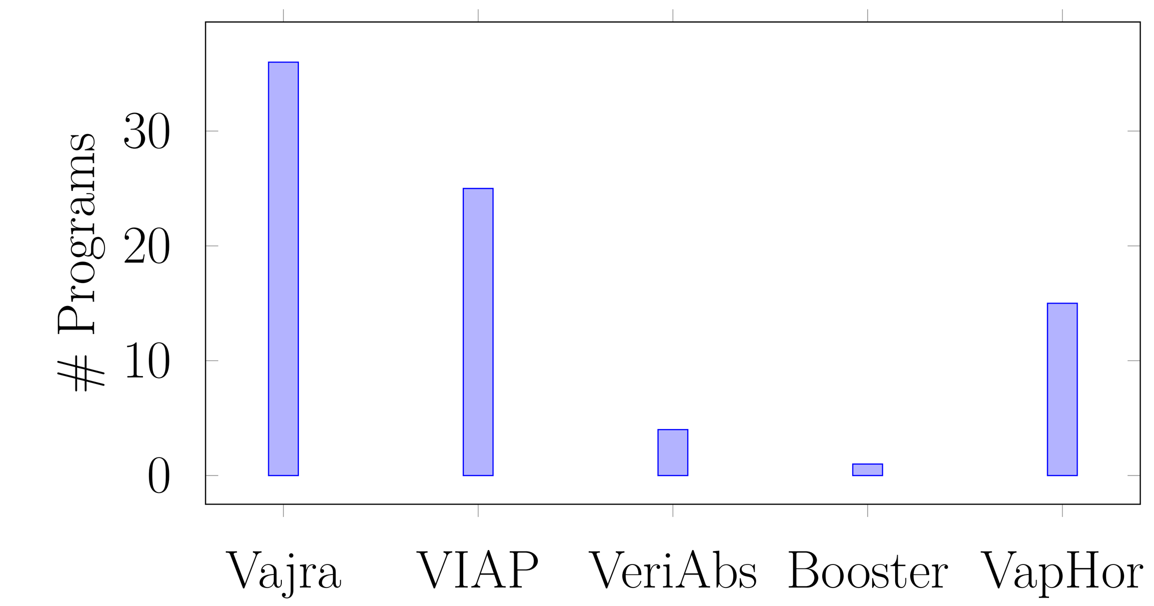
Inferring and Proving Pre's

```
assume(B[N-2]=(N-1)³-(N-2)³);

assume(∀i∈[0,N-1),C[i]=i³);
A[N-1] = A[N-2] + 6;
B[N-1] = B[N-2] + A[N-2];
C[N-1] = C[N-2] + B[N-2];
assert(C[N-1]=(N-1)³);

assert(B[N-1]=(N)³-(N-1)³);
```

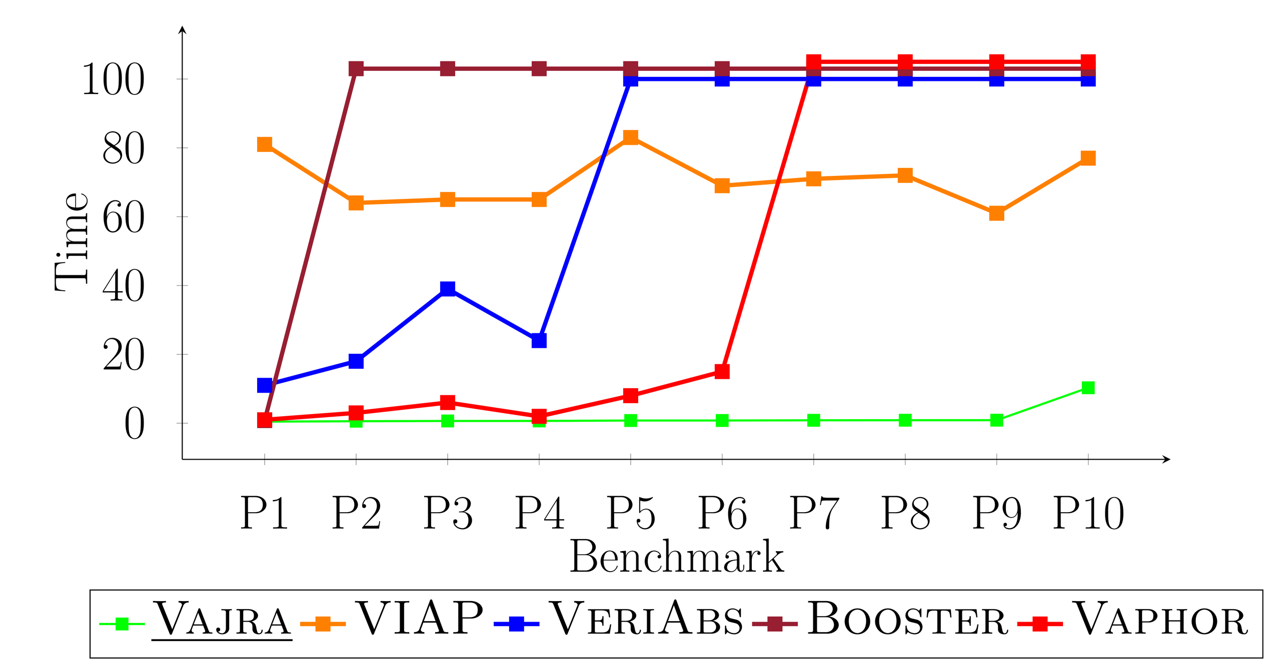
Vajra Tool Diagram



Efficiently Solves More Programs



Comparison of Execution Time of Tools



Recognition

- VAJRA is now a part of TCS tool - VERIABS
- Participated in the software verification competition SV-COMP 2020
- Won a gold medal in Reach-safety category
- Stood first in arrays sub-category

Conclusion

- *Difference makes the difference!*
- *Reduction to verification of loop free programs*
- *Compute differences of programs recursively*
- *Weakest pre-condition computation to infer new facts aiding induction*

References

[1] S. Chakraborty, A. Gupta, and D. Unadkat. Verifying array manipulating programs with full-program induction. In *Proc. of TACAS*, 2020.