

# Verifying Programs with Lists <sup>2</sup>

Peter Fontana, Piotr Mardziel

May 6, 2009

---

<sup>2</sup>... and maybe trees too

# Verification (all you need to know)

## Windows

A fatal exception 0E has occurred at 0028:C0011E36 in UXD UMM(01) + 00010E36. The current application will be terminated.

- \* Press any key to terminate the current application.
- \* Press CTRL+ALT+DEL again to restart your computer. You will lose any unsaved information in all applications.

Press any key to continue \_

# Verification (cont.)

- Describe desired program property and show it holds.
  - Proof by exhaustive enumeration not plausible.
  - Derive desired property via derivation rules / axioms.
    - First order logic derivation rules,
    - Hoare Logic derivation rules, and
    - Domain specific axioms: list reachability, integer (in)equalities, etc.

$$\begin{array}{c}
 \frac{\vdash A \quad \vdash B}{\vdash A \wedge B} \quad \frac{\vdash [a/x]A \quad (a \text{ is fresh})}{\vdash \forall x.A} \quad \frac{\vdash \forall x.A}{\vdash [E/x]A} \quad \frac{\vdash [a/x]A}{\vdash [a/x]A} \\
 \frac{\vdash A}{\vdash B} \quad \frac{\vdash A \Rightarrow B \quad \vdash A}{\vdash B} \quad \frac{\vdash [E/x]A}{\vdash \exists x.A} \quad \frac{\vdash \exists x.A \quad \dots}{\vdash B} \\
 \vdash A \Rightarrow B
 \end{array}$$

$$\frac{\vdash \{A\} \text{ skip } \{A\}}{\vdash \{A\} c_1 \{B\} \quad \vdash \{B\} c_2 \{C\}} \quad \vdash \{A\} c_1; c_2 \{C\} \\
 \frac{\vdash \{A \wedge b\} c_1 \{B\} \quad \vdash \{A \wedge \neg b\} c_2 \{B\}}{\vdash \{A\} \text{ if } b \text{ then } c_1 \text{ else } c_2 \{B\}}$$

$$\begin{array}{l}
 \forall x \quad . \quad x \rightsquigarrow x \\
 \forall x, y, z \quad . \quad x \rightsquigarrow y \wedge y \rightsquigarrow z \Rightarrow x \rightsquigarrow z \\
 \forall x \quad . \quad x \neq \perp \Rightarrow x \rightsquigarrow (x \rightarrow \text{next}) \\
 \forall x, y \quad . \quad x \rightsquigarrow y \Rightarrow x = y \vee (x \rightarrow \text{next}) \\
 \quad . \quad \rightsquigarrow y \\
 \forall x \quad . \quad \perp \rightsquigarrow x \Rightarrow x = \perp
 \end{array}$$

# Annotation

```
InsertionSort(Array A, int n) a
1  i := 1;
2  while (i < n)
     $\forall y : (i \leq y < n)$ 
     $\Rightarrow (\tilde{A}[y] = A[y]) \wedge$ 
3     $\forall y \exists x : (0 \leq y < i)$ 
     $\Rightarrow (\tilde{A}[y] = A[x] \wedge 0 \leq x < i)$ 
4    j := i - 1; val := A[i];
5    while (j ≥ 0 ∧ A[j] > val)
        val =  $\tilde{A}[j] \wedge -1 \leq j < i \wedge$ 
6         $\forall y : (i < y < n) \Rightarrow \tilde{A}[y] = A[y] \wedge$ 
         $\forall y \exists x : (0 \leq y < i)$ 
         $\Rightarrow (\tilde{A}[y] = A[x] \wedge 0 \leq x \leq i \wedge x \neq j + 1)$ 
7        A[j + 1] := A[j];
8        j := j - 1;
9        A[j + 1] := val;
10       i := i + 1;
11  Assert( $\forall y \exists x : (0 \leq y < n) \Rightarrow (\tilde{A}[y] = A[x] \wedge 0 \leq x < n)$ )
```

<sup>a</sup>Example from [SG09]

- Require programmer annotations
  - ▶ Desired program property,
  - ▶ Loop invariants,
  - ▶ Preconditions
- A lot of work,
- Invariants / preconditions might not be obvious

# System Overview (Saurabh's VS<sup>3</sup> [SGF09])

- **Goal:**

- ▶ Given the program *postcondition* (user-specified via `Assert`),
- ▶ Compute the weakest precondition needed for the program to satisfy the *postcondition*.

- But the Program needs the user's help! Give the program *templates* and *predicates*

- **Templates:** A Skeleton sketch of the format of the *loop invariants* and *program preconditions*.

- **Predicates:** A set of relations/properties that specify the search-space of constraints.

## List Axioms from [SGF09]

We use  $x \rightsquigarrow y$  if informally, node  $x$  *leads to* node  $y$  or  $y$  is *reachable from*  $x$ .

Formally,  $\rightsquigarrow$  is a relation with the properties given below:

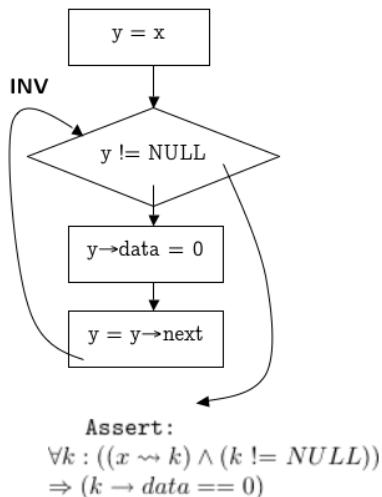
$\forall x$	$\cdot$	$x \rightsquigarrow x$	Reflexivity
$\forall x, y, z$	$\cdot$	$x \rightsquigarrow y \wedge y \rightsquigarrow z \Rightarrow x \rightsquigarrow z$	Transitivity
$\forall x$	$\cdot$	$x \neq \perp \Rightarrow x \rightsquigarrow (x \rightarrow \text{next})$	Step: Head
$\forall x, y$	$\cdot$	$x \rightsquigarrow y \Rightarrow x = y \vee (x \rightarrow \text{next}) \rightsquigarrow y$	Step: Tail
$\forall x$	$\cdot$	$\perp \rightsquigarrow x \Rightarrow x = \perp$	End

(Here,  $\perp = \text{NULL}$ )

## A Few System Details

- Consider the Initial Precondition and the Invariants unknowns
- Use the Hoare Logic Rules to generate implications that involve these unknowns
- With these implications, we can set certain predicates to true or false
- Use the List Axioms to help obtain predicate forms from the constraints
- Then use an SMT Solver to find a solution (satisfiable assignment for each of the predicates).

## Applying to listinit.c



- **Predicates:**  $(x \rightsquigarrow k)$ ,  $(x \rightsquigarrow y)$ ,  $(y \not\rightsquigarrow k)$
- **Template:**  $\forall k : k \rightarrow \text{data} == 0$ .
- From Hoare Logic Rules:  
**INV**  $\wedge (y == \text{NULL}) \Rightarrow$   
 $(\forall k : ((x \rightsquigarrow k) \wedge (k \neq \text{NULL})) \Rightarrow (k \rightarrow \text{data} == 0))$
- From the above constraint, the predicate  $x \rightsquigarrow k$  must be *true* in **INV**.
- The Axioms allow us to work with the  $k \neq \text{NULL}$  in the postcondition.

Figure: Initializing a List - the CFG of `listinit.c`

# Goals

- ✓ Develop sufficient understanding of VS<sup>3</sup>.
- (Potentially) extend axioms of [SGF09] to sufficiently infer preconditions/invariants for verification of various benchmarks in [BR06].
  - ▶ List-Reverse, Sorted-Insert, Bubble-Sort, Remove-Elements (filter)
- (Hopefully) determine a set of axioms for trees and use tool for various tree-based algorithms.
  - ▶ Binary search tree operations

# Questions?



# References

 Jesse D. Bingham and Zvonimir Rakamaric.

A logic and decision procedure for predicate abstraction of heap-manipulating programs.

In *VMCAI*, pages 207–221, 2006.

 Saurabh Srivastava and Sumit Gulwani.

Program verification using templates over predicate abstraction.

In *PLDI '09: Proceedings of the 2009 ACM SIGPLAN conference on Programming language design and implementation*, 2009.

 Saurabh Srivastava, Sumit Gulwani, and Jeffrey S. Foster.

Vs3: Smt solvers for program verification.

In *CAV '09: Proceedings of Computer Aided Verification 2009*, 2009.