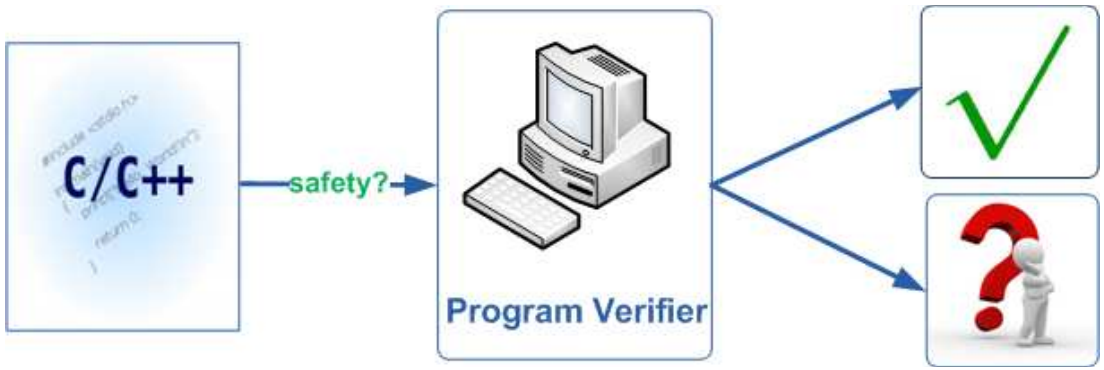
Towards Complete Specifications with an Error Calculus

Le Quang Loc
National University of Singapore

Joint work with Asankhaya Sharma, Florin Craciun and Wei-Ngan Chin

Student Session at POPL2013
January 26, 2013

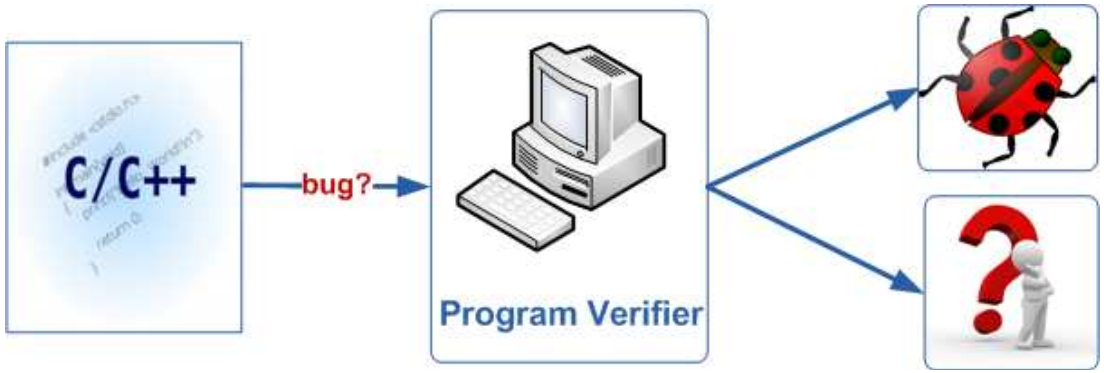
Motivation: Finding Bugs and Proving Safety



Program verification system:

- typically used for proving safety
- may produce false positives

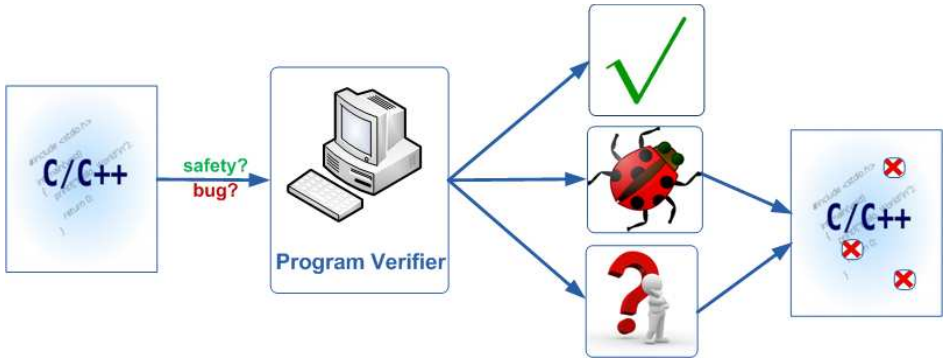
Motivation: Finding Bugs and Proving Safety



Program verification system:

- additionally used for bugs finding.
- may not have a definite answer, in general.

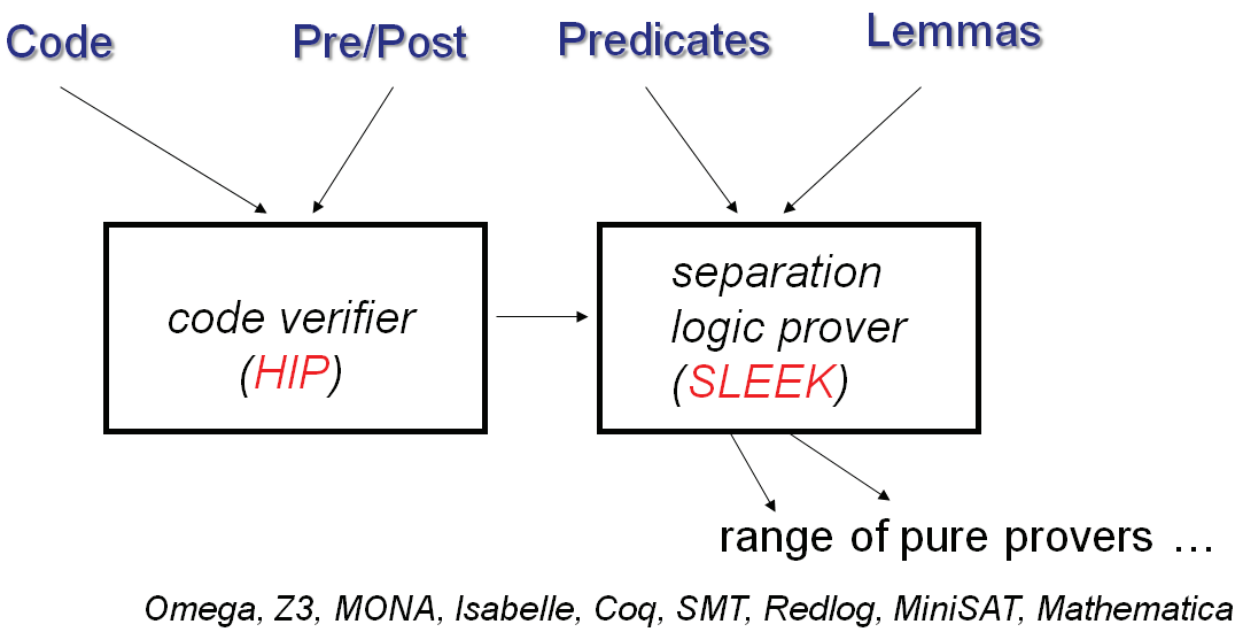
Our goal



In this work, we present

1. an automatic *modular* verification system
2. for safety proving and bugs finding
3. and further assist users through errors localization.
4. on heap-manipulating programs through separation logic.

HIP/SLEEK

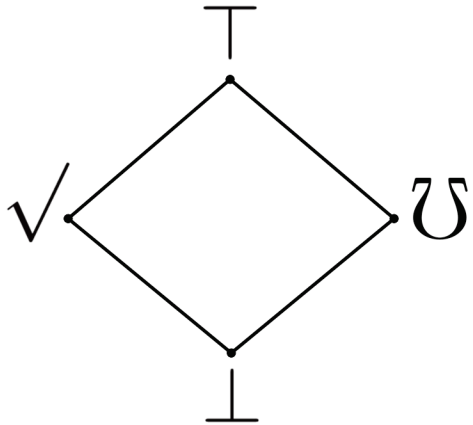


settings:

- symbolic execution
- safety verification: exit points and function call points.
- modularity: function call is compositionally verified through pre/post

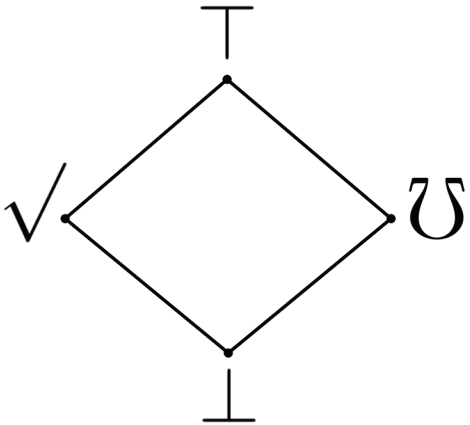
Key idea 1: The Lattice

The status of program states are defined as first class values.



unreachability:	$\neg\texttt{sat}(\pi_1)$	$x{\geq}0 \wedge x{<}0 \wedge \texttt{res}{=}{-}1 \vdash_C \texttt{res}{>}0 \rightsquigarrow \bot$
safety:	$\texttt{valid}(\pi_1 \vdash \pi_2)$	$x{\geq}0 \wedge y{>}1 \wedge \texttt{res}{=}1 \vdash_C \texttt{res}{>}0 \rightsquigarrow \surd$
must error:	$\texttt{sat}(\pi_1)$ *and*	$x{\geq}0 \wedge \neg(y{>}1) \wedge y{<}0 \wedge \texttt{res}{=}{-}1$
	$\texttt{valid}(\pi_1 \vdash \neg\pi_2)$	$\vdash_C \texttt{res}{>}0 \rightsquigarrow ℧$
may error:	$\texttt{sat}(\pi_1)$	$x{\geq}0 \wedge \neg(y{>}1) \wedge \neg(y{<}0) \wedge \texttt{res}{=}y$
	$\texttt{sat}(\pi_1 \vdash \pi_2)$	$\vdash_C \texttt{res}{>}0 \rightsquigarrow \top$
	$\texttt{sat}(\pi_1 \vdash \neg\pi_2)$	

Key idea 1: The Lattice




```
1   int foo(int x, int y)
2    requires x≥0
3    ensures (res>0)√;{
4     if (x<0) return −1; /*L1*/
5     else {
6      if (y>1) return 1; /*L2*/
7      else if (y<0) return −1;/*L3*/
8       else return y; /*L4*/
9    }
10  }
```

Key idea 2: More Complete Specification

data structure:

```
data node {
 int val;   node next;
}
```

- Traditional Approach:

```
int get_data(node x)
 requires x↦node⟨d, p⟩ ensures (res=d);
{
 return x.val;
}
```

- Our Proposal:

```
int get_data(node x)
 case{ x≠null →
        requires x↦node⟨d, p⟩ ensures (res=d) √;
       x=null → ensures (true ) Ʊ; }
```

Key idea 3: Enhance Separation Entailment Procedure

$$\Delta_1 \vdash \Delta_2 * \Delta_R \qquad \Delta_1 \vdash \Delta_2 \rightsquigarrow (\Delta_R, \tau)$$

Example:

$$x{\mapsto}\mathtt{node}(\_, q) * q{\mapsto}\mathtt{node}(\_, \mathtt{null}) \vdash x{\mapsto}\mathtt{node}(\_, \mathtt{null})$$

- Old Entailment Procedure:

$$(q{\neq}\mathtt{null} \wedge x{\neq}\mathtt{null} \vdash q{=}\mathtt{null} : \mathit{fails})$$

- Enhanced Entailment Procedure:
  1. $q{\neq}\mathtt{null} \wedge x{\neq}\mathtt{null} \vdash q{=}\mathtt{null}$: $\tau_p{=}\mho$
  2. $\textsc{sat}(\mathbf{x}{\mapsto}\mathtt{node}(\_, \mathbf{q}) * \mathbf{q}{\mapsto}\mathtt{node}(\_, \mathtt{null}))$

$$x{\mapsto}\mathtt{node}(\_, q) * q{\mapsto}\mathtt{node}(\_, \mathtt{null}) \vdash x{\mapsto}\mathtt{node}(\_, \mathtt{null}) \rightsquigarrow (..., \mho)$$

Key idea 4: The Calculus with Operators

$\sqcap$	$\top$	υ	$\surd$	$\bot$
$\top$	$\top$	υ	$\surd$	$\bot$
υ	υ	υ	$\bot$	$\bot$
$\surd$	$\surd$	$\bot$	$\surd$	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

$\circ$	$\top$	υ	$\surd$	$\bot$
$\top$	$\top$	υ	$\top$	$\bot$
υ	υ	υ	υ	$\bot$
$\surd$	$\top$	υ	$\surd$	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

$\cup$	$\top$	υ	$\surd$	$\bot$
$\top$	$\top$	$\top$	$\surd$	$\bot$
υ	$\top$	υ	$\surd$	$\bot$
$\surd$	$\surd$	$\surd$	$\surd$	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

Motivation of the operators on error statuses:

- more complex entailment:
  1. $\Delta_1 \vee \Delta_2 \vdash \Delta_3 \vee \Delta_4$
  2. proof search with lemmas
- localize the ”slice” caused the must/may errors:

$$x{>}3 \wedge y{>}7 \ \vdash \ x{<}0 \wedge y{>}3 \rightsquigarrow \upsilon$$

- verification with case specifications

Summary

- Technical details and Experiments can be found in the technical report at `http://www.comp.nus.edu.sg/~locle/`
- Futher Study
  - `case` in postconditions
  - CEGAR for separation logic
    - Diagnosing Abstraction Failure for Separation Logic-based Analyses (Josh Berdine et.al. - CAV2012)
    - Abstraction Refinementfor Separation Logic Program Analyses (Stephen Magill et. al. - online access)

Questions?

Thank you!