A deductive verification tool for realistic programs

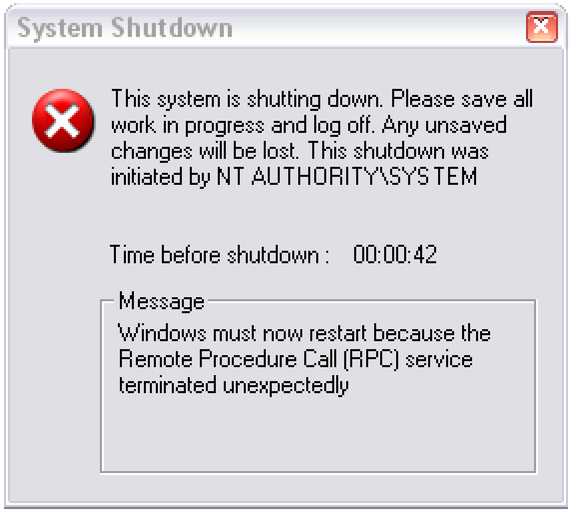
ROSEAC 2010 Workshop @ Jeju
POSTECH Programming Language Laboratory
Jonghyun Park

Introduction

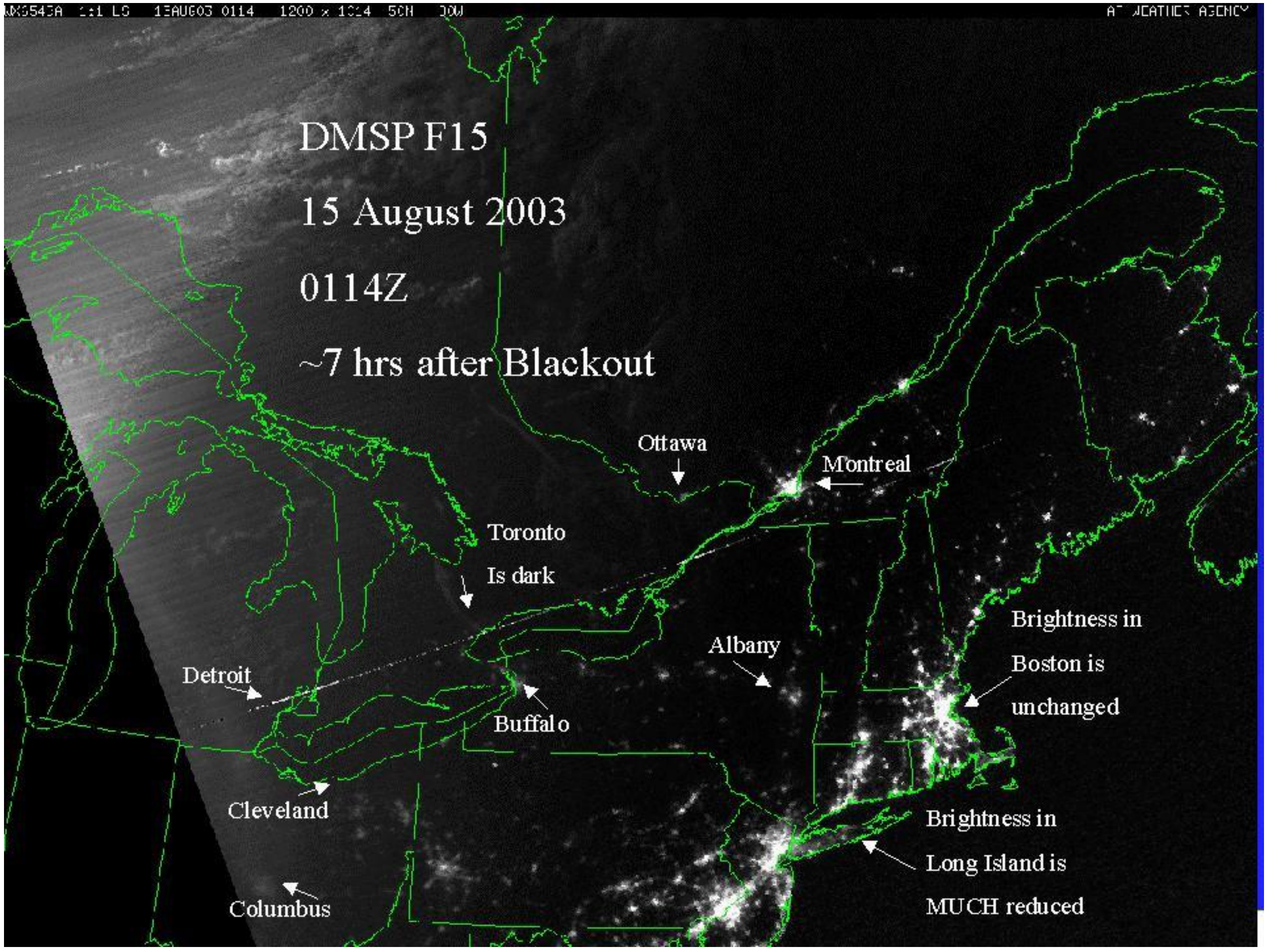
Arian 5 (1996) - $500 million

Orbiter (1999) - $125 million

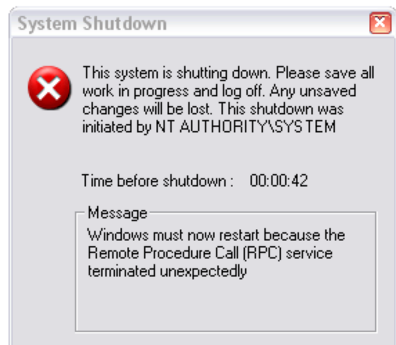
Blaster (2003) - $1.3 billion



Blackout (2003) - $6 billion



Program verification is important!

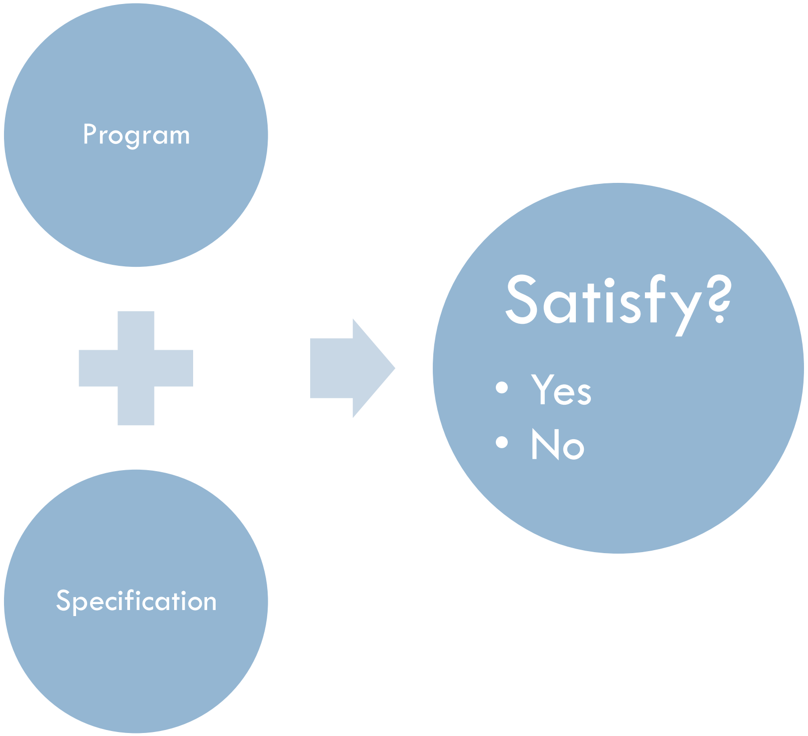


Techniques for program verification

- Testing
- Abstract interpretation
- Model checking
- Deductive verification
- ……

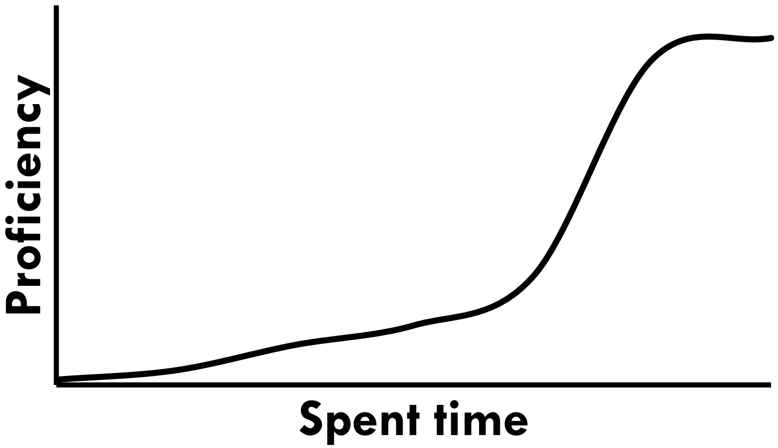
Deductive verification?

- A program verification technique using theorem proving



Problem?

- Difficult specification



- Lots of user effort

Then, why?

- Can formalize and prove far-reaching properties of programs
- Can model the semantics of program language precisely
  - No abstraction from unbounded data structures

Especially…

- For some small software, we need to prove complex properties very precisely
  - A garbage collector only collects <u>unused</u> objects?
  - A device driver always returns a <u>valid</u> value for every request?
- Important for embedded software!
  - A controller always sends a control signal for <u>physically possible</u> actions?

Goal

- Develop a deductive verification tool for realistic programs!
  - Support pointers, dynamic allocations, recursive data structures, ……
  - Less user effort!

Previous work?

JESSIE

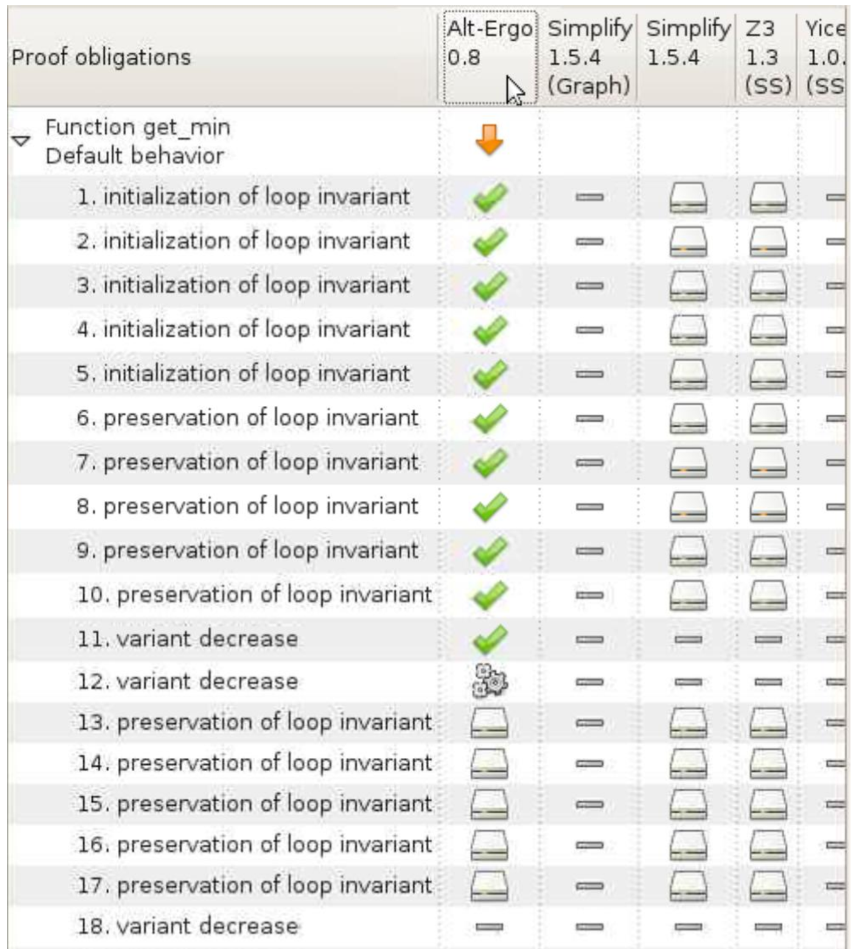
- A plug-in for deductive verification in Frama-C based on Hoare Logic [1]
- Prove that C functions satisfy specification as expressed in ACSL
- Automation!
  - Automatic annotation generation
  - Automatic proving by external tools
- Support pointer, dynamic allocation, recursive data structures, ……

An example: JESSIE

```
/*@ predicate min_over_array(int *arr, int len, int min) =
  @   \forall integer i; 0 <= i < len ==> arr[min] <= arr[i];
  @*/

/*@ requires 0 < len && \valid_range(arr,0,len-1);
  @ ensures 0 <= \result < len;
  @ ensures min_over_array(arr,len,\result);
  @*/
int get_min(int* arr, int len) {
  int min = 0;
  /*@ loop invariant 0 <= i <= len && 0 <= min < len;
    @ loop invariant min_over_array(arr,i,min);
    @ loop variant len - i;
    @*/
  for (int i = 0; i < len; ++i) {
    if (arr[i] < arr[min]) { min = i; }
  }
  return min;
}
```

An example: JESSIE



JESSIE is good, but

Problem?

- Memory assertion?
- Complex proof!
  - Handling alias [2]
- Solution?
  - Separation Logic!

Separation Logic

- An extension of *Hoare Logic* by John C. Reynolds [3] with separating connectives
- Allow specification about heap
- Capture the insight of informal argument by "Local reasoning"

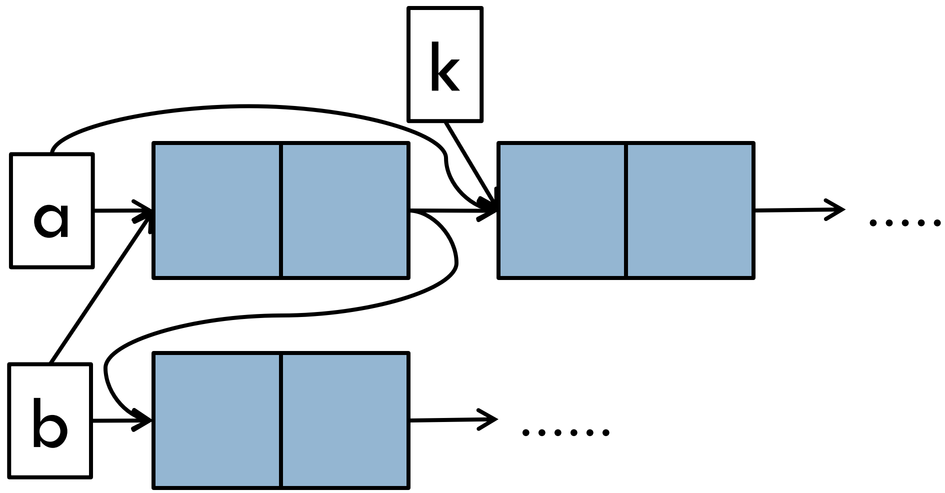
An example: Memory assertion

```
{ … }
int* create_int_cell() {
    …
}
{ ∃n : int. \ret → n }
```

Does *create_int_cell* only allocate memory for an integer?

An example: List reverse

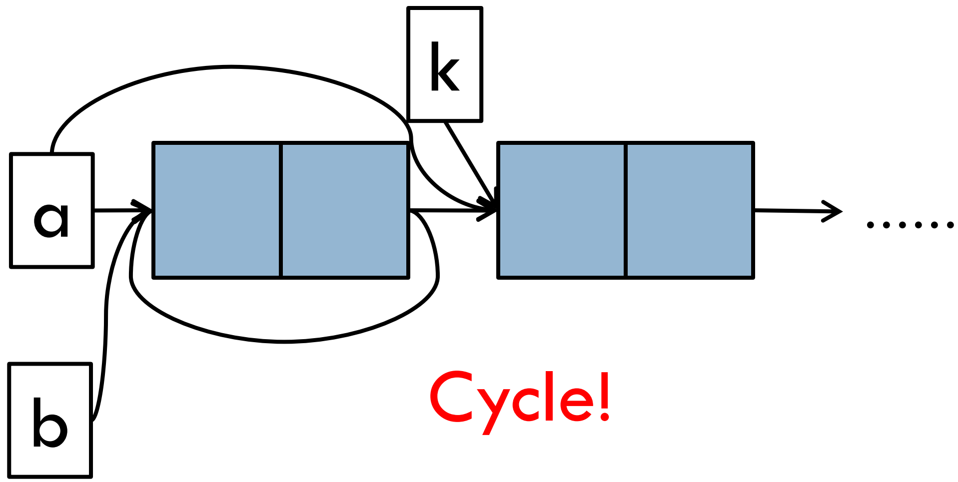
```
b := nil
while a != nil do
    k := [a + 1];
    [a + 1] := b;
    b := a;
    a := k;
end while
```



Reverse (hd::tl) l = Reverse tl (hd::l)

Loop invariant:
No sharing between a, b

```
while a != nil do
    k := [a + 1];
    [a + 1] := b;
    b := a;
    a := k;
end while
```



Hoare Logic vs. Separation Logic

```
while a != nil do
   k := [a + 1];
   [a + 1] := b;
   b := a;
   a := k;
end while
```

Hoare Logic:

$(\exists \alpha, \beta.\ \mathrm{List}\ \alpha\ a \wedge \mathrm{List}\ \beta\ b \wedge \alpha_0^R = \alpha^R \cdot \beta) \wedge$
$(\forall k.\ \mathrm{Reach}(a, k) \wedge \mathrm{Reach}(b, k) \Rightarrow k = \mathrm{nil})$

Separation Logic:

$(\exists \alpha, \beta.\ \mathrm{List}\ \alpha\ a * \mathrm{List}\ \beta\ b \wedge \alpha_0^R = \alpha^R \cdot \beta)$

Hoare Logic vs. Separation Logic

```
while a != nil do
    k := [a + 1];
    [a + 1] := b;
    b := a;
    a := k;
end while
```

Hoare Logic:

$(\exists \alpha, \beta.\ \text{List}\ \alpha\ a \wedge \text{List}\ \beta\ b \wedge \alpha_0{}^R = \alpha^R \cdot \beta) \wedge \text{List}\ \gamma\ x$

$(\forall k.\ \text{Reach}(a, k) \wedge \text{Reach}(b, k) \Rightarrow k = \text{nil}) \wedge$

$(\forall k.\ \text{Reach}(x, k) \wedge (\text{Reach}(a, k) \vee \text{Reach}(b, k)) \Rightarrow k = \text{nil})$

Separation Logic:

$(\exists \alpha, \beta.\ \text{List}\ \alpha\ a * \text{List}\ \beta\ b * \text{List}\ \gamma\ x \wedge \alpha_0{}^R = \alpha^R \cdot \beta)$

Frame rule!

$(\exists \alpha, \beta.\ \text{List}\ \alpha\ a * \text{List}\ \beta\ b \wedge \alpha_0{}^R = \alpha^R \cdot \beta)$

What happens there exists another list x unrelated to list a, b?

Goal (refined)

- Develop a deductive verification tool for realistic programs
  - Use the idea of <u>Separation Logic</u>

Related works

- Interactive program verification with Separation Logic [4, 5]
  - Embed separation logic in existing interactive theorem prover such as Coq and HOL/Isabelle
- Automated program verification with Separation Logic
  - Support limited data structures such as linked lists and trees [6]
  - Support limited form of specifications [7]

Currently……

- Automation!
  - Develop a theoretic foundation for automated proving using the idea of Separation Logic, which is a model of Boolean BI [8]

Current roadmap

Cut-free sequent calculus for BI [9]

Contraction-free sequent calculus for weak BI

Cut-free sequent calculus for Boolean BI

Cut-free sequent calculus for Boolean BI-like Logic

A variant of Separation Logic for program verification

Cut-free contraction-free sequent calculus for weak BI

$$\frac{A \text{ atomic}}{A \Longrightarrow A}\ Init \qquad \frac{A \text{ atomic}}{\Delta; A \Longrightarrow A}\ Init'$$

$$\frac{\delta(\Delta) \Longrightarrow C}{\delta(\Delta, \mathsf{I}) \Longrightarrow C}\ \mathsf{W}' \qquad \frac{\delta(\Delta) \Longrightarrow C}{\delta(\Delta, (\mathsf{I}; \Sigma)) \Longrightarrow C}\ \mathsf{W}''$$

$$\frac{}{\Delta \Longrightarrow \top}\ \top R \qquad \frac{}{\delta(\bot) \Longrightarrow C}\ \bot L$$

$$\frac{\Delta; A \supset B \Longrightarrow A \quad \delta(\Delta; A \supset B; B) \Longrightarrow C}{\delta(\Delta; A \supset B) \Longrightarrow C}\ \supset L \qquad \frac{\Delta; A \Longrightarrow B}{\Delta \Longrightarrow A \supset B}\ \supset R$$

$$\frac{\delta(A; B; A \wedge B) \Longrightarrow C}{\delta(A \wedge B) \Longrightarrow C}\ \wedge L \qquad \frac{\Delta \Longrightarrow A \quad \Delta \Longrightarrow B}{\Delta \Longrightarrow A \wedge B}\ \wedge R$$

$$\frac{\delta(A; A \vee B) \Longrightarrow C \quad \delta(B; A \vee B) \Longrightarrow C}{\delta(A \vee B) \Longrightarrow C}\ \vee L \qquad \frac{\Delta \Longrightarrow A}{\Delta \Longrightarrow A \vee B}\ \vee R_L \qquad \frac{\Delta \Longrightarrow B}{\Delta \Longrightarrow A \vee B}\ \vee R_R$$

$$\frac{\Delta \Longrightarrow A \quad \delta(\Delta', B) \Longrightarrow C}{\delta(\mathsf{WC}[\Delta, \Delta', A \mathrel{-\!\star} B]) \Longrightarrow C}\ {-\!\star} L \qquad \frac{\mathsf{I} \Longrightarrow A \quad \delta(\Delta, B) \Longrightarrow C}{\delta(\mathsf{WC}[\Delta, A \mathrel{-\!\star} B]) \Longrightarrow C}\ {-\!\star} L' \qquad \frac{\Delta \Longrightarrow A \quad \delta(B) \Longrightarrow C}{\delta(\mathsf{WC}[\Delta, A \mathrel{-\!\star} B]) \Longrightarrow C}\ {-\!\star} L''$$

$$\frac{\Delta, A \Longrightarrow B}{\Delta \Longrightarrow A \mathrel{-\!\star} B}\ {-\!\star} R \qquad \frac{\delta(A, B) \Longrightarrow C}{\delta(A \star B) \Longrightarrow C}\ \star L$$

$$\frac{\Delta \Longrightarrow A \quad \Delta' \Longrightarrow B}{\mathsf{WC}[\Delta, \Delta'] \Longrightarrow A \star B}\ \star R \qquad \frac{\mathsf{I} \Longrightarrow A \quad \Delta \Longrightarrow B}{\Delta \Longrightarrow A \star B}\ \star R' \qquad \frac{\Delta \Longrightarrow A \quad \mathsf{I} \Longrightarrow B}{\Delta \Longrightarrow A \star B}\ \star R''$$

Cut-free sequent calculus for Boolean BI-like logic

$$\frac{A \text{ atomic}}{\omega[A \longrightarrow_{\mathsf{B}} A]}\ \mathit{Init}$$

$$\frac{\omega[\Delta \longrightarrow_{\mathsf{B}} \Psi]}{\omega[\Delta; \Delta' \longrightarrow_{\mathsf{B}} \Psi]}\ \mathsf{W} \quad \frac{\omega[\Delta \longrightarrow_{\mathsf{B}} \Psi]}{\omega[\Delta \longrightarrow_{\mathsf{B}} \Psi; A]}\ \mathsf{W}' \quad \frac{\omega[\Delta; \Delta'; \Delta' \longrightarrow_{\mathsf{B}} \Psi]}{\omega[\Delta; \Delta' \longrightarrow_{\mathsf{B}} \Psi]}\ \mathsf{C} \quad \frac{\omega[\Delta \longrightarrow_{\mathsf{B}} \Psi; A; A]}{\omega[\Delta \longrightarrow_{\mathsf{B}} \Psi; A]}\ \mathsf{C}'$$

$$\frac{}{\omega[\bot \longrightarrow_{\mathsf{B}} \cdot]}\ \bot L \quad \frac{\omega[\Delta \longrightarrow_{\mathsf{B}} \Psi]}{\omega[\Delta \longrightarrow_{\mathsf{B}} \Psi; \bot]}\ \bot R \quad \frac{\omega[\Delta \longrightarrow_{\mathsf{B}} A; \Psi]}{\omega[\Delta; \neg A \longrightarrow_{\mathsf{B}} \Psi]}\ \neg L \quad \frac{\omega[\Delta; A \longrightarrow_{\mathsf{B}} \Psi]}{\omega[\Delta \longrightarrow_{\mathsf{B}} \neg A; \Psi]}\ \neg R$$

$$\frac{\omega[\Delta; A; B \longrightarrow_{\mathsf{B}} \Psi]}{\omega[\Delta; A \wedge B \longrightarrow_{\mathsf{B}} \Psi]}\ \wedge L \quad \frac{\omega[\Delta \longrightarrow_{\mathsf{B}} A; \Psi] \quad \omega[\Delta \longrightarrow_{\mathsf{B}} B; \Psi']}{\omega[\Delta \longrightarrow_{\mathsf{B}} A \wedge B; \Psi; \Psi']}\ \wedge R$$

$$\frac{\omega[\Delta; \emptyset_{\mathsf{m}} \longrightarrow_{\mathsf{B}} \Psi]}{\omega[\Delta; \mathsf{I} \longrightarrow_{\mathsf{B}} \Psi]}\ \mathsf{I}L \quad \frac{}{\omega[\emptyset_{\mathsf{m}} \longrightarrow_{\mathsf{B}} \mathsf{I}]}\ \mathsf{I}R$$

$$\frac{\omega[(\Delta' \longrightarrow_{\mathsf{B}} \Psi'; A), (\Delta \longrightarrow_{\mathsf{B}} \Psi); \Delta'' \longrightarrow_{\mathsf{B}} \Psi''] \quad \omega[B; \Delta'' \longrightarrow_{\mathsf{B}} \Psi'']}{\omega[(\Delta' \longrightarrow_{\mathsf{B}} \Psi'), (\Delta; A \mathrel{-\!\!\star} B \longrightarrow_{\mathsf{B}} \Psi); \Delta'' \longrightarrow_{\mathsf{B}} \Psi'']}\ \mathrel{-\!\!\star} L$$

$$\frac{(\Delta \longrightarrow_{\mathsf{B}} \Psi), (A \longrightarrow_{\mathsf{B}} \cdot) \longrightarrow_{\mathsf{B}} B}{\omega[(\Delta \longrightarrow_{\mathsf{B}} A \mathrel{-\!\!\star} B; \Psi), (\Delta' \longrightarrow_{\mathsf{B}} \Psi'); \Delta'' \longrightarrow_{\mathsf{B}} \Psi'']}\ \mathrel{-\!\!\star} R$$

$$\frac{\omega[\Delta; (A \longrightarrow_{\mathsf{B}} \cdot), (B \longrightarrow_{\mathsf{B}} \cdot) \longrightarrow_{\mathsf{B}} \Psi]}{\omega[\Delta; A \star B \longrightarrow_{\mathsf{B}} \Psi]}\ \star L$$

$$\frac{\omega[\Delta''; (\Delta \longrightarrow_{\mathsf{B}} \Psi; A), (\Delta' \longrightarrow_{\mathsf{B}} \Psi') \longrightarrow_{\mathsf{B}} \Psi''] \quad \omega[\Delta''; (\Delta \longrightarrow_{\mathsf{B}} \Psi), (\Delta' \longrightarrow_{\mathsf{B}} \Psi'; B) \longrightarrow_{\mathsf{B}} \Psi'']}{\omega[\Delta''; (\Delta \longrightarrow_{\mathsf{B}} \Psi), (\Delta' \longrightarrow_{\mathsf{B}} \Psi') \longrightarrow_{\mathsf{B}} A \star B; \Psi'']}\ \star R$$

Theorem 2.2 (cut elimination).

If $\omega[\Delta \longrightarrow_{\mathsf{B}} A; \Psi]$ *and* $\omega[\Delta'; A \longrightarrow_{\mathsf{B}} \Psi']$, *then* $\omega[\Delta; \Delta' \longrightarrow_{\mathsf{B}} \Psi; \Psi']$.

Question?

Reference

1. Multi-prover verification of C programs, Filliâtre J.C and Marché, C.
2. Proving pointer programs in Hoare logic, Bornat, R.
3. Separation logic: A logic for shared mutable data structures, Reynolds, J.C.
4. Types, bytes, and separation logic, Tuch, H. and Klein, G. and Norrish, M
5. Practical Tactics for Sepration Logic, McCreight, A.
6. Smallfoot: Modular automatic assertion checking with separation logic, Berdine, J. and Calcagno, C. and O'Hearn P.W.

Reference

7. Separation Logic Verication of C Programs with an SMT Solver, Matko, B. and Matthew, P. and Wolfram, S.
8. BI as an assertion language for mutable data structures, Ishtiaq, S.S. and O'Hearn, P.W.
9. The logic of bunched implications, O’Hearn, P.W. and Pym, D.J.