In this talk:

- background on pointers
- separation logic
Why Reason about Pointers?

Pointers are everywhere!

- operating system kernels (Linux)
- device drivers
- network code (TCP/IP)
- web servers (Apache)
- anything involving C/C++
- even Java and ML have references
The Problem with Pointers

\{ \text{valid } p \land \text{valid } q \} \quad *q = 42; \quad *p = 7; \quad \{ *p = 7 \land *q = ? \}
The Problem with Pointers

$\{ \text{valid } p \land \text{valid } q \}$

$q = 42;$

$p = 7;$

$\{ p = 7 \land q = ? \}$

\[ \Rightarrow *q = 42 \]

\[ \Rightarrow *q = 7 \]
Some Simpler Approaches
**A Simple Plan**

```plaintext
datatype ref = Ref int | Null

types heap = int ⇒ val

datatype val = Int int | Bool bool | Struct x int int bool | . . .
```

- hp :: heap, p :: ref
- pointer access: *p = the_Int (hp (the_addr p))
- pointer update: *p := v = hp := hp ((the_addr p) := v)

- a bit klunky
- gets worse with structs
- lots of value extraction (the_Int) in spec and program
A linked list struct with next pointer and element:

\[
\begin{align*}
\text{datatype} & \quad \text{ref} = \text{Ref int} \mid \text{Null} \\
\text{types} & \quad \text{next}_{hp} = \text{int} \Rightarrow \text{ref} \\
\text{types} & \quad \text{elem}_{hp} = \text{int} \Rightarrow \text{int}
\end{align*}
\]

- next :: next_{hp}, elem :: elem_{hp}, p :: ref
- pointer access: p->next = next (the_addr p)
- pointer update: p->next := v = next := next ((the_addr p) := v)

- a separate heap for each struct field
- p->next and p->elem can't alias
- assumes a type-safe language
- p1->next and p2->next can still alias
Separation Logic
The heap represents computer memory

- partial map: allocated and unallocated regions
- emp: a heap with no allocated regions
- we'll use a simple version based on natural numbers
- and steal 0 to mean the null pointer
Separating Conjunction

Primary mechanism of separation logic

- assign resources (e.g. heap) to predicates
- predicates consume resources
- no resource sharing across separating conjunction

\[ h_0 \perp h_1 \equiv \text{dom } h_0 \cap \text{dom } h_1 = \{\} \]

\[(P \land^* Q) h \equiv \exists h_0 \ h_1 . \ h = h_0 ++ h_1 \land h_0 \perp h_1 \land P \ h_0 \land Q \ h_1 \]
Benefits of Local Reasoning

\[ f(p_s) \]

\[ p \downarrow \]

\[ p \downarrow \]

\[ s \rightarrow s' \]
Benefits of Local Reasoning

\[ Q \]

\[ f \ p \ s \]

\[ s' \]
Benefits of Local Reasoning

\[ Q \xrightarrow{f} p \xrightarrow{s} Q \]

\[ s \xrightarrow{p} s' \]
Benefits of Local Reasoning

\[
\{ p \mapsto - \} \quad f \quad p \quad s \quad \{ p \mapsto \textcolor{red}{\text{Red}} \}
\]
Benefits of Local Reasoning

\[
\{ p \rightarrow - \land^* Q \} \quad f \quad p \quad s \quad \{ p \rightarrow \land^* Q \}
\]
\[
\begin{align*}
\{ p \leftrightarrow - \} \quad f \quad p \quad s \quad \{ p \leftrightarrow \quad \quad \} \\
\{ p \leftrightarrow - \wedge^* Q \} \quad f \quad p \quad s \quad \{ p \leftrightarrow \quad \wedge^* Q \} 
\end{align*}
\]
Benefits of Local Reasoning

The Frame Rule

\[
\begin{align*}
\{ P \} & \quad stmt & \quad \{ P' \} \\
\{ P \land^* Q \} & \quad stmt & \quad \{ P' \land^* Q \}
\end{align*}
\]
Precise Mapping Predicates

The maps-to predicate defines a heap

- with only one valid pointer
- combine with other mappings to make bigger heaps
- remember to use separating conjunction!

\[(p \mapsto v) \ h \equiv (h \ p = \text{Some } v \land \text{dom } h = \{p\})\]

\[(p \mapsto -) \equiv \exists v. \ (p \mapsto v)\]
Defining a Linked List

Demo
What's old:
- local variables used for calculations
- the usual constructs: SKIP, IF, WHILE, ";;"
- and local variable assignment
- with identical Hoare rules

What's new:
- a variable representing the heap
- want precise specification of assignment to pointer
- need a way to allocate/free memory
Allocation rule:

\[ \{ \text{emp} \} \text{ alloc } x \ [e_1, e_2, \ldots, e_n] \ \{ \ x \leftrightarrow e_1 \land^* \ldots\land^* \ x + n \leftrightarrow e_n \ \} \]

Disposal rule:

\[ \{ \ x \leftrightarrow - \ \} \text{ dispose } x \ \{ \ \text{emp} \ \} \]
The normal, local assignment rule:

\[
\{ x \leftrightarrow - \} [x] := v \{ x \leftrightarrow v \}
\]

Using the magic wand (separating implication):

\[
(P \rightarrow^* Q) h \equiv \forall h'. h' \perp h \land P h' \rightarrow Q (h ++ h')
\]

we can make it a backwards-reasoning rule:

\[
\{ x \leftrightarrow - \land^* (x \leftrightarrow v \rightarrow^* P) \} [x] := v \{ P \}
\]
Demo
Conclusion

Separation Logic

- is a nice way to reason about pointers
- doesn't need specification of what doesn't change
NICTA

From imagination to impact