Automating Separation Logic Reasoning

• Separation Logic is a bad fit for SMT solvers
  • Predicates are higher-order
  • Predicates are often recursive
  • Relies on Associative-Commutative (AC) reasoning
    \[ P \sqcap Q \sqcap R \Leftrightarrow R \sqcap P \sqcap Q \]

• Automation through a cooperation between SMT solving and custom separation logic decision procedures
A Syntax-Directed Frame Rule

• **Problem:** Applications of the frame rule are non-deterministic

\[
\Gamma \vdash c : \{Q\} t \{R\} \\
\Gamma \vdash c : \{? P \star Q\} t \{? P \star R\}
\]

• **Solution:** Deterministically apply framing at the “leaf” only, during function calls

\[
\Gamma \vdash v : a. \quad \Gamma \vdash f : a \rightarrow \{Q\} t \{R\} \\
\Gamma \vdash f \, v : \{? P \star Q\} t \{? P \star R\}
\]
Automating Frame Inference: An Example

val write (r:ref a) (x:a) : Steel unit (ptr r) (ptr r)
let two_writes (r1 r2:ref int) : Steel unit (ptr r1 ⋆ ptr r2) (ptr r1 ⋆ ptr r2)
    = write r1 0; // : {?F1 ⋆ ptr r1} unit {?F1 ⋆ ptr r1}
    write r1 1 // : {?F2 ⋆ ptr r1} unit {?F2 ⋆ ptr r1}

• Observation: Separation logic VCAs can be seen as AC-unification problems
  for instance, ptr r1 ⋆ ptr r2 ⇐ {?F1} ⋆ ptr r1

• Observation: A scheduling of equivalences where each problem contains at most one metavariable exists
  ptr r1 ⋆ ptr r2 ⇐ {?F1} ⋆ ptr r1,
  ?F1 ⋆ ptr r1 ⇐ {?F2} ⋆ ptr r1
  ?F2 ⋆ ptr r1 ⇐ ptr r1 ⋆ ptr r2
Solving Frame Metavariables

We reduced the problem to solving equivalences of the shape

\(?F \star P1 \star P2 \iff Q1 \star Q2\)

We provide a decision procedure for these problems as an F* tactic, which:

• Supports existentially quantified ghost variables
• Can query the SMT solver for equalities on subterms
• Sacrifices completeness for speed and user interaction
Steel Example: Spinlocks

```plaintext
val lock (p:slprop) : Type

val acquire (l:lock p) : Steel unit emp (\_ \to p)

val release (l:lock p) : Steel unit p (\_ \to emp)
```

Initially, no ownership

Transferring ownership

Transferring ownership back
Steel Example: Invariants ("Ghost locks")

val inv (p:slprop) : Type

Invariants can be accessed inside **atomic commands**

Composition of a physical action and a finite number of ghost operations

This enables lock-free shared-memory concurrency
Steel Example: Michael-Scott 2-lock queues

- The shape of the queue is captured by an invariant
- The dequeuer and the enqueuer both have a lock on the head and tail pointers respectively
Steel Example: Message-Passing Concurrency

val chan : Type
val endpoint (ch:chan) (p:prot) : slprop

Returns a new channel

val new (p:prot) : Steel chan emp (λc → endpoint c p ★ endpoint c (dual p))

Returns permissions for two parties to use the new channel for protocol p

At this stage of the protocol, we must send a message

val send (c:chan{is_send_next next}) (x:msg_t next) :
Steel unit (endpoint c next) (λ_ → endpoint c (step next x))

The message x is compatible with the current state of the protocol

val recv ...

We initially are at the stage next of the protocol on channel c

val close ...

After executing send, we advanced the state of the protocol by sending x
Steel Example: PingPong Protocol

```plaintext
let pingpong : prot =
    let x = Protocol.send int in
    let y = Protocol.recv (y:int{y > x}) in
    Protocol.done

let client (c:chan) : Steel unit (endpoint c pingpong) (\_ \to emp)
  = send c 17;
    let y = recv c in
    assert (y > 17);
    close ()
```

Statically checked assert, erased at runtime
Separating Separation and First-Order Logic

• We associate to each separation logic predicate a *self-framing selector*
  For example, a reference’s selector is the value it contains

• First-order logic predicates about *selectors* can be discharged by SMT

```plaintext
val swap (p1 p2:ref int) : Steel unit (ptr p1 ★ ptr p2) (ptr p1 ★ ptr p2)
  (requires λ _ → T)
  (ensures λ s0 _ s1 → s0.[p1] == s1.[p2] ∧ s0.[p2] == s1.[p1])
```
Steel Example: Binary Trees

type node a = {data : a; left : t a; right: t a}
and t a = ref (node a)

val tree (ptr:t a) : slprop

let rec height (ptr:t a) : Steel int (tree ptr) (λ _ → tree ptr)

(requires λ _ → T) The contents of the tree are unchanged

(ensures λs0 x s1 → s0.[ptr] == s1.[ptr] ∧ Spec.height s0.[ptr] == x)

= if is_null ptr then (unroll_leaf ptr; 0) else (
let node = unroll_tree ptr in
let hleft = height node.left in
let hright = height node.right in roll_tree ptr node.left node.right;
if hleft > hright then (hleft + 1) else (hright + 1))
Steel Example: AVL Trees

type node a = {data : a; left : t a; right: t a}
and t a = ref (node a)

val tree (ptr:t a) : slprop

val insert_avl (cmp:Spec.cmp a) (ptr:t a) (v:a)
: Steel (t a) (tree ptr) (λ ptr’ → tree ptr’)

(requires λ s → Spec.is_avl cmp s.[ptr])

(ensures λs0 ptr’ s1 → Spec.is_avl cmp s1.[ptr] ∧
 s1.[ptr’] == Spec.insert_avl cmp s0.[ptr] v)

The AVL invariant is preserved

Same abstract predicate as before

Functional correctness