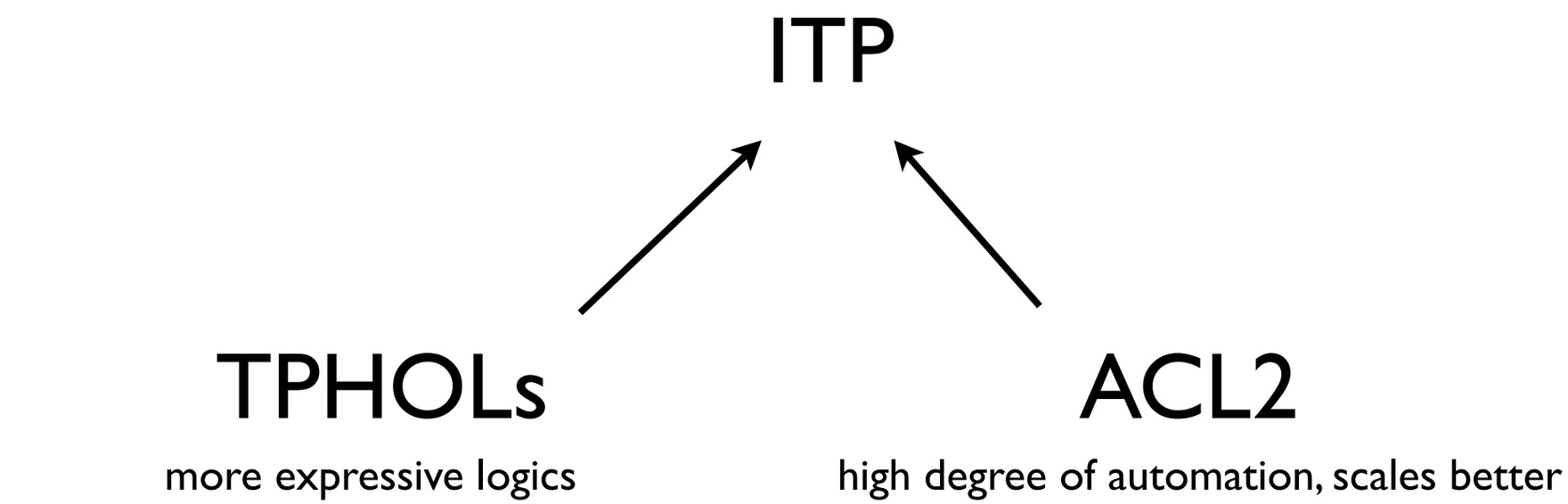
Separation logic adapted for proofs by rewriting

Magnus O. Myreen
University of Cambridge

short paper at ITP'10

Motivation



In this talk: separation logic adapted to ACL2-like proofs (rewriting)

Separation logic in one slide

- An extension to Hoare logic due to Reynolds et al. (~2002)
- Its separating conjunction ($*$) prevents pointer aliasing:

$$(a \mapsto b) * (a{+}1 \mapsto x) * (b \mapsto 0) * (b{+}1 \mapsto y)$$

- Its frame rule makes reasoning local:

$$\{p\}\ c\ \{q\} \implies \forall r.\ \{p * r\}\ c\ \{q * r\}$$

Problematic quantifiers

- Definition of separating conjunction:

$$(p * q)\ s \quad = \quad \exists s_1\ s_2.\ (s = s_1 \uplus s_2) \wedge p\ s_1 \wedge q\ s_2$$

- Quantifiers also in frame rule, linked-list predicate, etc.

Avoiding quantifiers

- Wrote an interpreter for $*$-separated predicates

$((a \mapsto x) * (b \mapsto y) * (c \mapsto z))\ state$

separate $[(a, x), (b, y), (c, z)]\ []\ state$

- The linked-list example and frame:

separate $([(a, b), (a{+}1, x), (b, 0), (b{+}1, y)] +\!\!+ frame)\ []\ state$

Avoiding quantifiers

- Wrote an interpreter for $*$-separated predicates

$((a \mapsto x) * (b \mapsto y) * (c \mapsto z))\ state$

separate $[(a, x), (b, y), (c, z)]\ []\ state$

$$\begin{array}{lcl} \text{separate}\ []\ t\ state & = & \text{all\_distinct}\ t \\ \text{separate}\ ((a, x)::l)\ t\ state & = & (state(a) = x) \wedge \text{separate}\ l\ (a::t)\ state \end{array}$$

- The linked-list example and frame:

separate $([(a, b), (a{+}1, x), (b, 0), (b{+}1, y)] +\!\!+ frame)\ []\ state$

Powerful proof automation

- Example: destructive list reversal
  - rewriting alone can automatically prove body of loop:

list p1 $(x :: xs) *$ list p2 (ys)

```
p3 = p1->tail;
p1->tail = p2;
p2 = p1;
p1 = p3;
```

list p1 $(xs) *$ list p2 $(x :: ys)$

- toy language where pc, code, regs are kept in memory (potential for pointer aliasing)

Powerful proof automation

- Example: destructive list reversal
  - rewriting alone can automatically prove body of loop:

separate (llist 1 $(x::xs)$ ++ llist 2 ys ++ $frame$ ++ ...) [3] $state$

list p1 $(x :: xs)$ * list p2 (ys)

```
p3 = p1->tail;
p1->tail = p2;
p2 = p1;
p1 = p3;
```

list p1 (xs) * list p2 $(x :: ys)$

separate (llist 1 xs ++ llist 2 $(x::ys)$ ++ $frame$ ++ ...) [3] $state$

  - toy language where pc, code, regs are kept in memory (potential for pointer aliasing)

Powerful proof automation

- Example: destructive list reversal
  - rewriting alone can automatically prove body of loop:

separate (llist 1 $(x::xs)$ ++ llist 2 ys ++ $frame$ ++ ...) [3] $state$

list p1 $(x :: xs)$ * list p2 (ys)

```
p3 = p1->tail;          mem[3] := mem[mem[1]];
p1->tail = p2;          mem[mem[1]] := mem[2];
p2 = p1;                mem[2] := mem[1];
p1 = p3;                mem[1] := mem[3]
```

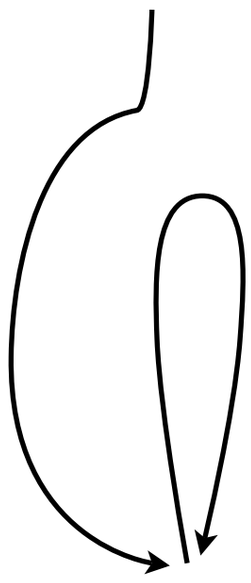
list p1 (xs) * list p2 $(x :: ys)$

separate (llist 1 xs ++ llist 2 $(x::ys)$ ++ $frame$ ++ ...) [3] $state$

  - toy language where pc, code, regs are kept in memory (potential for pointer aliasing)

Verified example

separate (llist 1 xs ++ $frame$ ++ ...) $[2, 3]$ $state$



```
 0:  mem[2] := 0;
 3:  jump to 18;
 6:  mem[3] := mem[mem[1]];
 9:  mem[mem[1]] := mem[2];
12:  mem[2] := mem[1];
15:  mem[1] := mem[3];
18:  jump to 6, if not (mem[1] = 0)
```

separate (llist 2 (reverse xs) ++ $frame$ ++ ...) $[1, 3]$ $state$

Summary

- Rewriting = powerful automation for separation logic if quantifiers are avoided

- Lesson learnt: HOL4's simplifier expands outermost match, ACL2's simplifier expands innermost match.

$$next(next(next(state)))$$

Ack. Matt Kaufmann ported my HOL4 implementation into ACL2.