Interactive Software Verification

Spring Term 2013

Holger Gast

gasth@in.tum.de

2.7.2013

Today: Introduction to Separation Logic

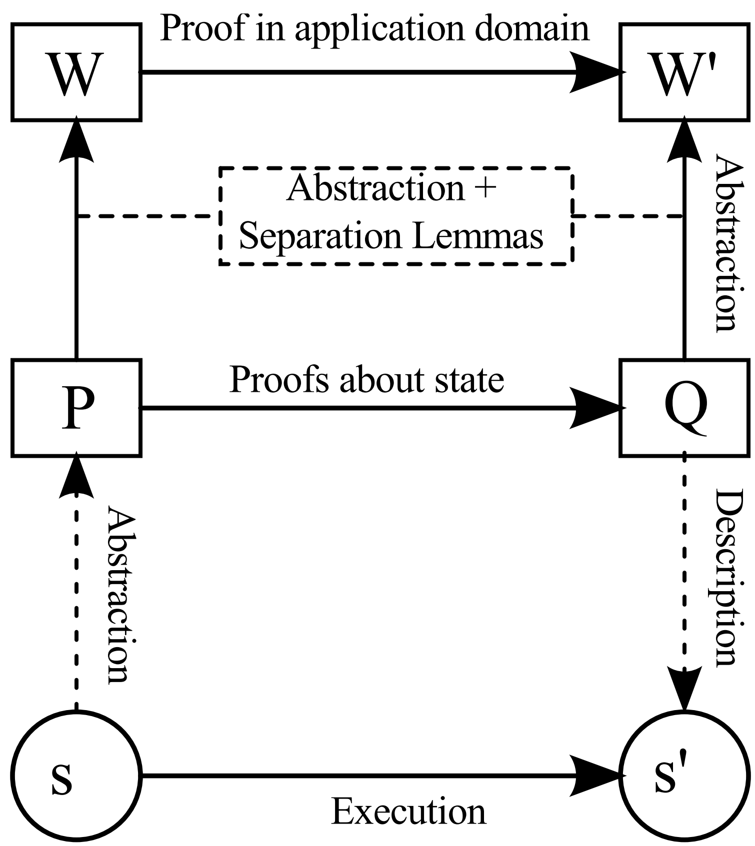
- Limits of Burstall's memory model
- Basics of Separation Logic [13, 16]
- Next week: verifying list algorithms

Recap: Loop Invariants ➪ List Insert

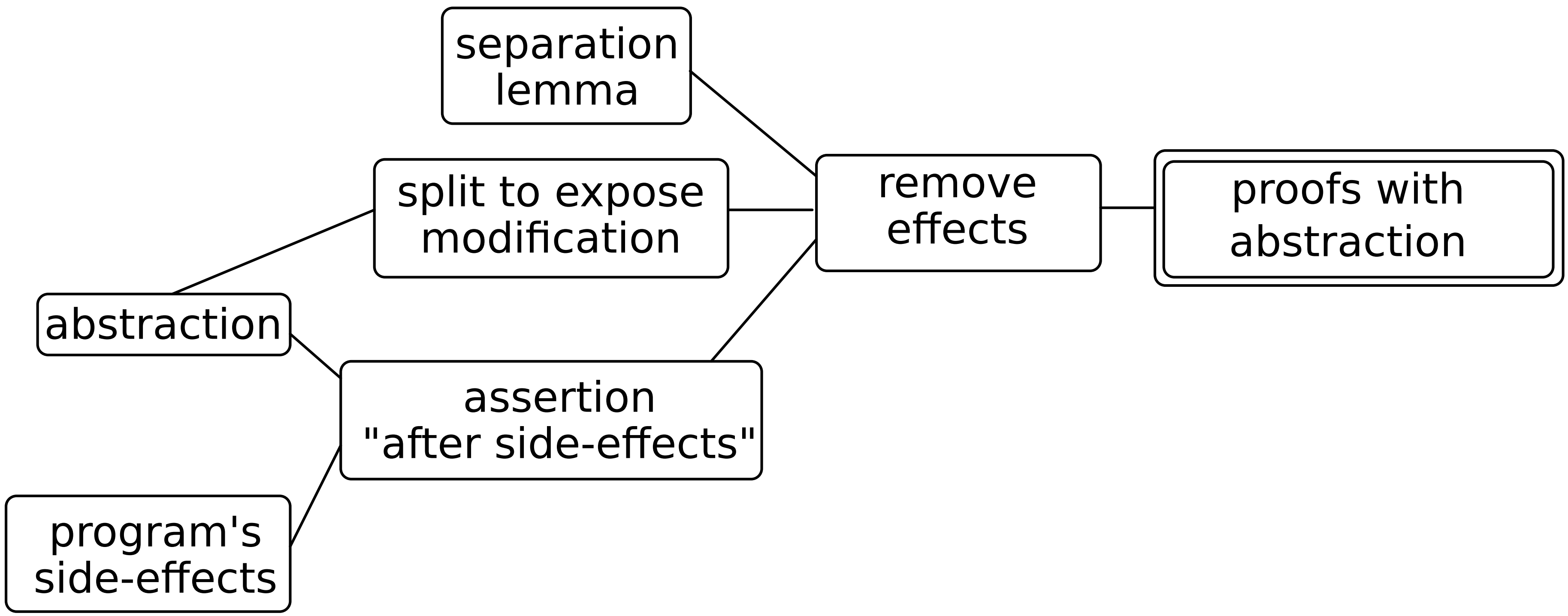
- Generalize the desired post-condition
  - Loop test is $a \neq b$, then post-condition from $a = b \wedge I$
  - If loop test is $a < b$, then add $a \leq b$ to invariant
- Describe (precisely) the achieved partial result
  - Consider the variables that are modified in the body
  - And capture the relationship between them
  - . . . and the relationship to the input values (logical vars)
- Add safety assertions
  - About the range of index variables
  - About pointers not being `null`
- Preserve information from before loop

$\Rightarrow$ Apply to Exercise 8.3 (interactive)

Recap: Correctness & Abstraction



Recap: Cheat Sheet



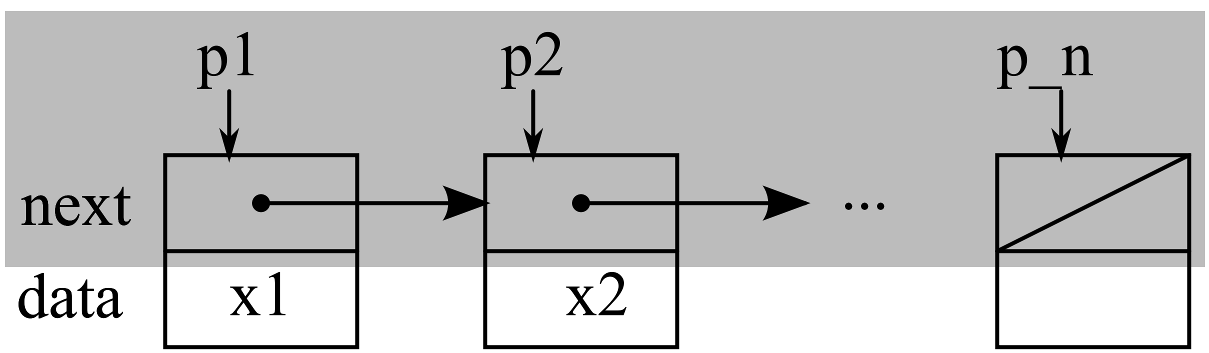
Limits of Burstall's Memory Model

Burstall's Memory Model

- The heap model
  - Introduce a separate array for each struct field
  - Index arrays by object pointers: $p \to \mathtt{f}$ becomes $\mathtt{f}[p]$
- Variant: explicit heap as two-dimensional array

$$p \to_h \mathtt{f} \text{ becomes } h\,(\mathtt{f}, p)$$

- Example: lists

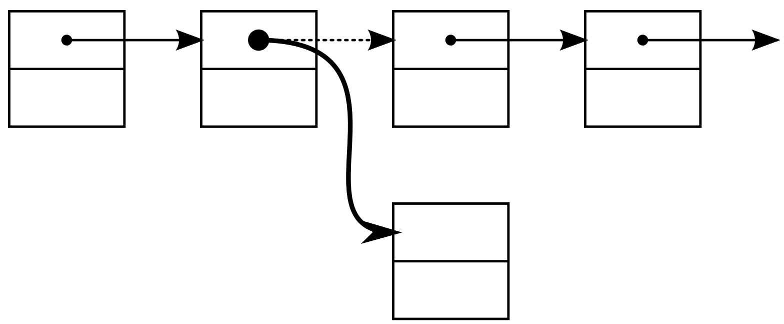


Linked Lists on the Heap

- Definition

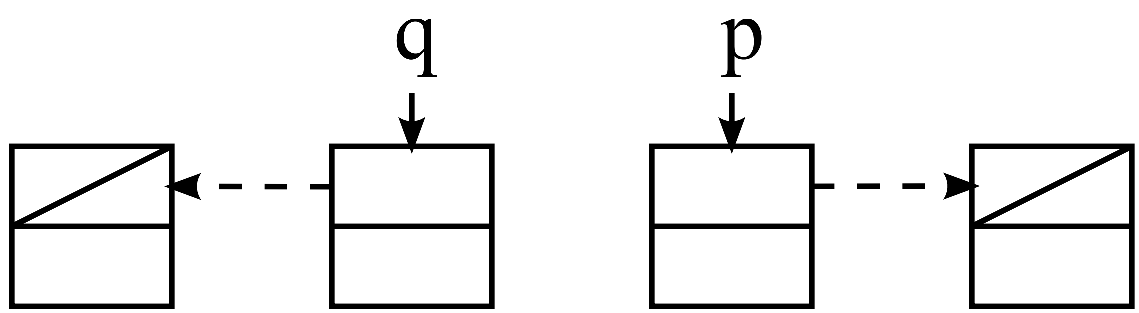
inductive list :: "alloc ⇒ nxt ⇒ addr ⇒ addr list ⇒ addr ⇒ bool"
where
emptyI: "$\llbracket$ p = q; xs = [] $\rrbracket \Longrightarrow$ list alloc nxt p xs q"
| consI: "$\llbracket$ alloc p; p ≠ NULL; list alloc nxt (field-get nxt p) xs' q
$\rrbracket \Longrightarrow$ list alloc nxt p (p # xs') q"

- Separation lemma



lemma list-sep[simp]:
"r ∉ set xs ⟹ list alloc (field-upd nxt r y) p xs q = list alloc nxt p xs q"

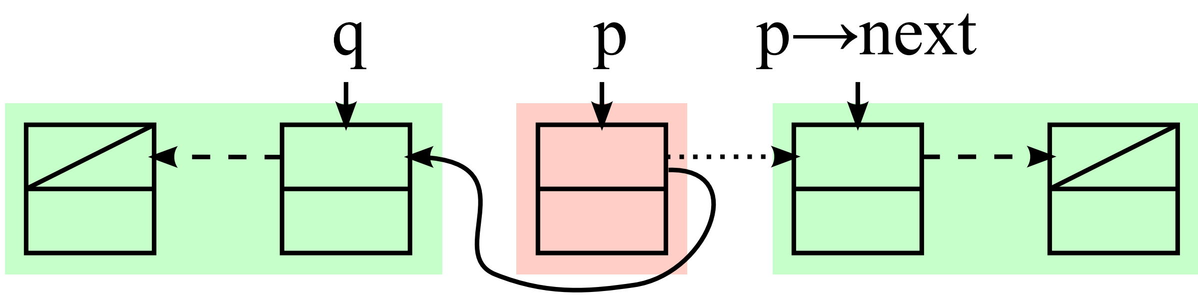
Example: Invariant of List Reversal



$$\exists\, \mathsf{ys}\ \mathsf{zs}.\ \mathsf{list\ alloc\ nxt\ p\ zs\ NULL} \wedge \mathsf{list\ alloc\ nxt\ q\ ys\ NULL} \wedge$$
$$\mathsf{XS} = \mathsf{rev\ ys} \mathbin{@} \mathsf{zs} \wedge \mathbf{set\ zs} \cap \mathbf{set\ ys} = \{\} \wedge \mathbf{distinct\ zs}$$

- Auxiliary variable `XS` for initial list
- Two partial lists starting at p and q
- So far `XS` has already been reversed partially
- Disjointness conditions to exclude aliasing

Disjointness in the Proof



$$\llbracket p \neq \mathrm{NULL};\ \underline{\mathrm{list\ alloc\ nxt}}\ q\ ys\ \mathrm{NULL};\ \mathrm{alloc}\ p;$$
$$\underline{\mathrm{list\ alloc\ nxt}}\ (\mathrm{field\text{-}get\ nxt}\ p)\ xs'\ \mathrm{NULL};$$
$$\mathbf{p \notin set\ ys;\ set\ xs' \cap set\ ys = \{\};\ p \notin set\ xs';\ distinct\ xs'}$$
$$\rrbracket \Longrightarrow \mathbf{list\ alloc\ (field\text{-}upd\ nxt\ p\ q)}\ (\mathrm{field\text{-}get\ nxt}\ p)\ xs'\ \mathrm{NULL} \wedge$$
$$\mathbf{list\ alloc\ (field\text{-}upd\ nxt\ p\ q)}\ q\ ys\ \mathrm{NULL} \wedge$$
$$\mathrm{set}\ xs' \cap \mathrm{set}\ (p \# ys) = \{\} \wedge \mathrm{rev}\ ys\ @\ p \# xs' = \mathrm{rev}\ (p \# ys)\ @\ xs'$$

lemma list-sep[simp]:
$$"\mathbf{r \notin set\ xs \Longrightarrow list\ alloc\ (field\text{-}upd\ nxt\ r\ y)}\ p\ xs\ q = \mathrm{list\ alloc\ nxt}\ p\ xs\ q"$$

Limits of Burstall's Model

- Specifications bloated with disjointness conditions
- Proof effort to exclude aliasing
  - Sets of addresses
  - $\Rightarrow$ Must axiomatize & automate set theory
- Abstraction predicates must yield addresses for sep. lemma
  - "Abstraction" is not very strong
  - Requires extra list-data function
- Supports limited range of language features [12]
  - No address operator (& in C)
  - No low-level, byte-addressed access (common in C)

Separation Logic

Separation Logic

- Idea: capture disjointness in syntactic structure of assertions
  - No need to mention addresses
  - No need for complex proofs in set theory
- Approach
  - State consists of heap and store (local variables)
  - Heap is mapping $\texttt{addr} \to \texttt{val}$ (generalization to bytes immediate)
  - Assertions are predicates on $(\texttt{heap} \times \texttt{store})$
- Introduce special connectives for assertions
  - $P \star Q$: P and Q hold on disjoint parts of the heap
  - $P -\!\!* Q$: Q holds on (disj.) extension of heap validating P
  - $p \mapsto v$: heap consists of single address p with value v

$\Rightarrow$ Capture heap layout & -content in simultaneously

A Success Story

- First proposed 2001/02 [13, 11]
- Applications to algorithms
  - Baum/DAG copies [3]
  - Schorr-Waite Graph Marking [17]
- Semi-Automatic proofs on fragment of symbolic heaps
  - Shape analysis [2, 6, 1]
  - Data structures (lists, etc.) [15]
  - Low-level C programs [4]
  - Garbage collectors [7]
- Interactive proofs
  - Memory manager [14]
  - Garbage collectors [9]

Our Goal

- Implement core of automatic methods [2, 15, 4]
- Retain Simpl as a basis [7]

⇒ Obtain straightforward proofs about list algorithms

```
pre: "(list p XS NULL) heap"
post: "(list ret (rev XS) NULL) heap"
q= null;
/*@inv
 (ℰ xs ys. (list p xs NULL ⋆ list q ys NULL) · XS = (rev ys @ xs)) heap
*/
while (p != null) {
 t = p→next;
 p→next = q;
 q = p; p = t;
}
```

- Demo: stepping through assignments
- Today: basics of separation logic

Basics in Isabelle: the Heap

- Make heap contain typed values (extension to bytes straightforward [7])

  datatype heap-val =
  VBool bool | VInt int | VNat nat | VPtr addr

- Heap is simply the expected mapping

  datatype addr = Addr nat
  datatype heap = Heap "addr $\rightharpoonup$ heap-val"

- Define accessors for domain and content

  hcnt (Heap c) = c
  hdom (Heap c) = dom c

- Program logic: introduce global variable `heap`

Values on the Heap

- A type descriptor packages the heap representation of values

  record 'a cty =
  ty-size :: "nat"
  ty-rep :: "'a ⇒ heap-val list"
  ty-val :: "heap-val list ⇒ 'a"

  - The size (i.e. consecutive heap addresses needed)
  - The representation function
  - The value function
- Only point of "encoding": might use byte sequences [7]
- Consistency of type descriptors

  ty-ok ct ≡ (∀ v. length (ty-rep ct v) = ty-size ct ∧
  ty-val ct (ty-rep ct v) = v)

⇒ Writing a value and then reading it yields the value

Accessing the Heap

- Only access heap through read, write, check-allocatedness
- Parametrize by type descriptor

$$\text{heap-get}\,\text{ct}\,\text{p}\,\text{h} \equiv \text{ty-val}\,\text{ct}\,(\text{map}\,(\text{the} \circ \text{hcnt}\,\text{h})\,(\text{addr-ivl}\,\text{p}\,(\text{ty-size}\,\text{ct})))$$

$$\begin{aligned}&\text{heap-put}\,\text{ct}\,\text{p}\,\text{v}\,\text{h} \equiv\\&\quad \text{Heap}\,(\text{hcnt}\,\text{h} +\!+ (\lambda \text{q}.\,\text{if}\,\text{q} \in \text{set}\,(\text{addr-ivl}\,\text{p}\,(\text{ty-size}\,\text{ct}))\\&\qquad\qquad\qquad\qquad \text{then Some}\,(\text{nth}\,(\text{ty-rep}\,\text{ct}\,\text{v})\,(\text{q} \ominus \text{p}))\\&\qquad\qquad\qquad\qquad \text{else None}))\end{aligned}$$

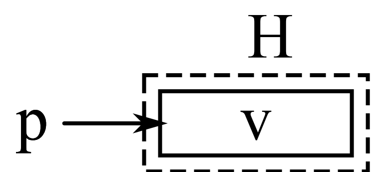
$$\begin{aligned}&\text{heap-acc-cond}\,(\text{ct} :: \text{'a cty})\,\text{p} \equiv\\&\quad \lambda \text{h}.\,\text{set}\,(\text{addr-ivl}\,\text{p}\,(\text{ty-size}\,\text{ct})) \subseteq \text{hdom}\,\text{h}\end{aligned}$$

- "put": if address is "overwritten", take local value
- Note: abstraction layer in theory

Points-To

- Points-to $p \mapsto v$ (or: singleton heap)
  - Heap contains v at address p
  - Heap allocates only addresses at p to store v

- Idea (H is the overall heap)

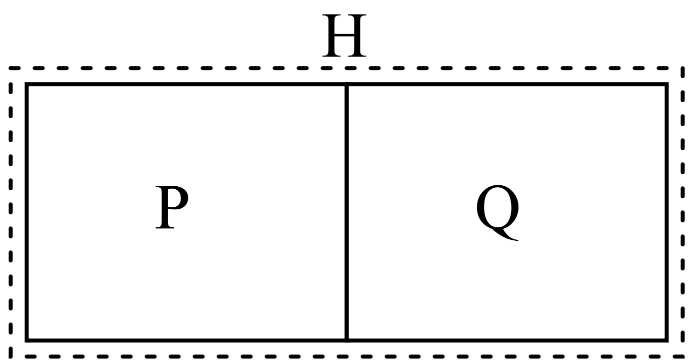


$$
\begin{aligned}
&\mathsf{p} \mapsto \mathsf{ct}.\,\mathsf{v} \equiv \\
&\quad \lambda \mathsf{h}.\ \mathsf{hdom}\ \mathsf{h} = \mathsf{set}\,(\mathsf{addr\text{-}ivl}\ \mathsf{p}\,(\mathsf{ty\text{-}size}\ \mathsf{ct})) \wedge \mathsf{heap\text{-}get}\ \mathsf{ct}\ \mathsf{p}\ \mathsf{h} = \mathsf{v} \wedge \mathsf{p} \neq \mathsf{NULL}
\end{aligned}
$$

$\Rightarrow$ Assertions also capture allocated addresses

Star

- $P \star Q$: spatial conjunction
- Idea (H is the overall heap)



- Definition ($\perp$ is "disjoint")

  $\mathsf{P} \star \mathsf{Q} \equiv \lambda \mathsf{h0}.\ \exists\ \mathsf{h}\ \mathsf{h'}.\ \mathsf{h} \perp \mathsf{h'} \wedge \mathsf{h0} = \mathsf{h} \uplus \mathsf{h'} \wedge \mathbf{P\ h} \wedge \mathbf{Q\ h'}$

- Meaning similar to $\wedge$, but with spatial side-conditions

Further Spatial Assertions

- `emp`: holds on empty heap
- `true`: holds for any heap
- Extend classical connectives to heaps
  - $P \wedge Q$: P and Q hold on the same heap
  - $P \vee Q$: analogously
  - $\forall x.\ P\ x$: P holds on heap for all values x
  - $\exists x.\ P\ x$: P holds on heap for some value x
- Pure assertions do not concern the heap

  ⇨ Capture local variables just as before

Defining Heap Predicates

- Heap predicates are simple: $\mathsf{hassn} = \mathsf{heap} \Rightarrow \mathsf{bool}$
- Infrastructure: predicates for structs

  structdef
  "struct node { int data; struct node *next; }"

$\Rightarrow$ Yields spatial predicate node p v, with HOL-record v

- Build lists inductively

  fun list :: "addr ⇒ Word list ⇒ addr ⇒ hassn"
  where
  "list p [] q = (emp · (p = q))"
  | "list p (x # xs') q = ($\mathcal{E}$ v. node p v ⋆ list (node-next v) xs' q
  · **x = node-data v**)"

- Operator "·": classical conjunction with pure assertion
- Note: xs are data values stored in list

Derived Connectives

- Value stored at address p

$$p \hookrightarrow v = p \mapsto v \star \texttt{true}$$

- "Some value": implicit existential quantification

$$p \mapsto - = \exists v.\, p \mapsto v$$

$$p \hookrightarrow - = \exists v.\, p \mapsto v \star \texttt{true}$$

- Records/structs

$$p \mapsto [v_1 \ldots v_n]$$

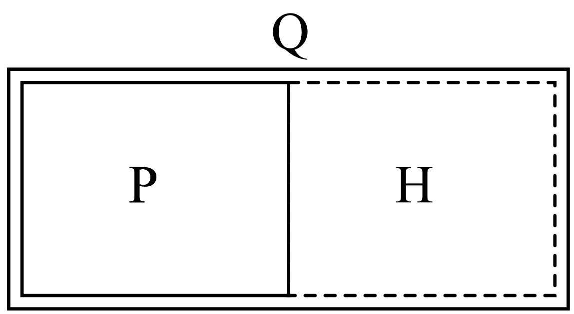
means

$$p \mapsto v_1 \ \star \cdots \star \ p \oplus (n-1) \mapsto v_n$$

connectives will not be further discussed

The "Magic Wand"

- $P \mathrel{-\!\!*} Q$: separating implication
  - If P holds on a disjoint extension of the current heap
  - Then Q holds for the extended heap
- Idea



- Def: $P \mathrel{-\!\!*} Q \equiv \lambda h.\, \exists h'.\, h \perp h' \wedge P\ h' \implies Q(h \uplus h')$
- Link to classical implication $P \longrightarrow Q$ (not formal, only for pedagogic purposes!)
  - Using additional assumption P
  - . . . statement Q holds

Application of Magic Wand

- Hoare rule for writing to the heap [13]

$$\{ \, (p \mapsto -) \star (p \mapsto e \mathrel{-\!\!*} Q) \, \} \; *p := e \; \{ \, Q \, \}$$

- Interaction with $\star$
  - Split heap Q into disjoint parts

  $\Rightarrow$ Extract the address to be modified

  ... and re-insert it with a new assumption
- Allocatedness: pre-condition asserts that p must be in the domain of the overall heap
- Likewise: reading from the heap (compare to Hoare's assignment axiom)

$$\{ \, \exists v. \, (p \mapsto v) \star (p \mapsto v \mathrel{-\!\!*} Q[v/x]) \, \} \; x := *p \; \{ \, Q \, \}$$

So far

- Basics of spatial assertions
- Can define heap predicates (also inductively)
- Attained main goals
  - Addresses hidden from abstraction predicates
  - Disjointness syntactically clear ⇨ no need for proofs
- Open issues
  - A Hoare logic for separation logic
  - Proofs within separation logic

Lemmas about Connectives

- Goal: proofs by syntactic formula manipulation
- Useful basic properties [13]

$$P \star (Q \star R) = (P \star Q) \star R$$

$$P \star Q = Q \star P$$

$$P \star \texttt{emp} = P$$

$$(P \vee Q) \star R = (P \star R) \vee (Q \star R)$$

$$(\exists x.\ P\ x) \star Q = \exists x.\ (P\ x \star Q)$$

- Wand & star

$$\frac{P \star Q \implies R}{P \implies (Q \mathrel{-\!\!*} R)} \qquad \frac{P \implies (Q \mathrel{-\!\!*} R)}{P \star Q \implies R}$$

Surprises

- We have [13]

$$(P \wedge Q) \star R \implies (P \star R) \wedge (Q \star R)$$

$$(\forall x.\ P\ x) \star Q \implies \forall x.\ (P\ x \star Q)$$

- But not the reverse directions
  - Address set of R is not fixed
  - Split of address set by $\star$ not fixed
  - Similar: different choices for x
- Unfortunately: no complete proof system [5]
- But: fragment without $-\!\!*$
  - Enables efficient proofs
  - Sufficient for many programs [2, 1, 10, 6, 15, 4, 7]

Hoare Logic

- Already seen: dereferencing

$$\{\ (p \mapsto -) \star (p \mapsto e \mathrel{-\!\!*} Q)\ \}\ \ *p := e\ \{\ Q\ \}$$

$$\{\ \exists v.\ (p \mapsto v) \star (p \mapsto v \mathrel{-\!\!*} Q[v/x])\ \}\ x := *p\ \{\ Q\ \}$$

- Allocation & deallocation

$$\{\ \forall v'.\ (v' \mapsto [\bar{e}]) \mathrel{-\!\!*} Q[v'/p]\ \}\ p := \mathtt{alloc}(\bar{e})\ \{\ Q\ \}$$

$$\{\ (p \mapsto -) \star Q\ \}\ \mathtt{dispose}(p)\ \{\ Q\ \}$$

$\Rightarrow$ All heap manipulation have concise rules

- Assignments to local variables: as before

The Frame Rule

$$\frac{\{\ P\ \}\ s\ \{\ Q\ \}}{\{\ P \star R\ \}\ s\ \{\ Q \star R\ \}}$$

if s does not change variables occurring in R

- Tight specifications: pre- and post-conditions mention only the used heap fragment, remainder implicitly unchanged
  - Hoare rules check for allocatedness

  $\Rightarrow$ Implicit statement about used fragment
- Special case: function call
  - Specification mentions accessed heap fragment
  - Remaining heap unmodified
- Note: also treat allocation/deallocation
- Refined version: fractional (read/write) permissions

Forward Reasoning

- Developers explain programs by mental execution (forward)
- But: classical Hoare logic works backwards
- Separation logic enables forward rules

$$\{\, p \mapsto - \star Q \,\} \; {*p} := e \; \{\, p \mapsto e \star Q \,\}$$

$$\{\, P \,\} \; p := \texttt{alloc}(\bar{e}) \; \{\, \exists p'.\, p \mapsto [\bar{e}] \star P[p'/p] \,\}$$

$$\{\, p \mapsto - \star P \,\} \; \texttt{dealloc}(p) \; \{\, P \,\}$$

- Application of such rules
  - Re-arrange given pre-condition to match rule
  - Apply rule
  - $\Rightarrow$ Obtain post-condition (final heap state)

$\Rightarrow$ Basis for automated methods

Local Reasoning

- Frame rule enables Local reasoning
  - Prove correctness first for heap actually used
  - Generalize to overall heap
- Examples: garbage collectors [8], Schorr-Waite [17]
- Can formulate alternative Hoare logic, e.g.

$$\{\, p \mapsto - \,\}\ *p := e\ \{\, p \mapsto e \,\}$$

- Benefits
  - Concentrate on essential statements about programs

  $\Rightarrow$ Smaller assertions in proofs
  - Possibly re-use proofs of repeated code

Summary

- Introduction to separation logic
  - Captures disjointness syntactically
  - True abstraction predicates (without internal addresses)
- Essential points
  - Spatial connectives for heaps

  $\Rightarrow$ Capture individual heap fragments & compose
  - Local reasoning
  - Tight specifications
  - Forward reasoning
- Next week: symbolic execution in separation logic

References

[1] J. Berdine, C. Calcagno, B. Cook, D. Distefano, P. O'Hearn, T. Wies, and H Yang. Shape analysis of composite data structures. In *CAV 2007*, volume 4590 of *LNCS*, Heidelberg, 2007. Springer.

[2] Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. Smallfoot: Modular automatic assertion checking with separation logic. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem P. de Roever, editors, *Formal Methods for Components and Objects, 4th International Symposium (FMCO)*, volume 4111 of *LNCS*. Springer, 2005.

[3] Richard Bornat, Cristiano Calcagno, and Peter O'Hearn. Local reasoning, separation and aliasing. In *Semantics, program analysis, and computing environments for memory management (SPACE'04)*, 2004.

[4] Matko Botincan, Matthew Parkinson, and Wolfram Schulte. Separation logic verification of c programs with an smt solver. In *4th International Workshop on Systems Software Verification (SSV)*, volume 254. Elsevier, 2009.

[5] Cristiano Calcagno, Hongseok Yang, and Peter W. O'Hearn. Computability and complexity results for a spatial assertion language for data structures. In *FST TCS '01: Proceedings of the 21st Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 108–119, London, UK, 2001. Springer.

[6] Dino Distefano, Peter W. O'Hearn, and Hongseok Yang. A local shape analysis based on separation logic. In Holger Hermanns and Jens Palsberg, editors, *TACAS*, volume 3920 of *LNCS*, pages 287–302. Springer, 2006.

[7] Holger Gast. Semi-automatic proofs about object graphs in separation logic. In Stephan Merz and Gerald Lüttgen, editors, *Automated Verification of Critical Systems (AVoCS '12)*, 2012.

[8] Andrew McCreight. *The Mechanized Verification of Garbage Collector Implementations*. PhD thesis, Department of Computer Science, Yale University, December 2008.

[9] Andrew McCreight. Practical tactics for separation logic. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, editors, *Theorem Proving in Higher Order Logics, 22nd International Conference (TPHOLs)*, volume 5674 of *LNCS*. Springer, 2009.

[10] Huu Hai Nguyen, Cristina David, Shengchao Qin, and Wei-Ngan Chin. Automated verification of shape and size properties via separation logic. In Byron Cook and Andreas Podelski, editors, *VMCAI*, volume 4349 of *LNCS*, pages 251–266. Springer, 2007.

[11] Peter W. O'Hearn, John C. Reynolds, and Hongseok Yang. Local reasoning about programs that alter data structures. In *Proceedings of the 15th International Workshop on Computer Science Logic*, number 2142 in LNCS, pages 1–19. Springer, 2001.

[12] Zvonimir Rakamaric and Alan J. Hu. A scalable memory model for low-level code. In Neil D. Jones and Markus Müller-Olm, editors, *Verification, Model Checking, and Abstract Interpretation, 10th International Conference (VMCAI 2009)*, volume 5403 of *LNCS*. Springer, 2009.

[13] John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science (LICS 02)*, 2002.

[14] Harvey Tuch, Gerwin Klein, and Michael Norrish. Types, bytes, and separation logic. In Martin Hofmann and Matthias Felleisen, editors, *Proc. 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'07)*, 2007.

[15] Thomas Tuerk. A formalisation of Smallfoot in HOL. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, editors, *Theorem Proving in Higher Order Logics, 22nd International Conference (TPHOLs)*, volume 5674 of *LNCS*. Springer, 2009.

[16] Tjark Weber. Towards mechanized program verification with separation logic. In Jerzy Marcinkowski and Andrzej Tarlecki, editors, *Computer Science Logic – 18th International Workshop, CSL 2004*, volume 3210 of *LNCS*, pages 250–264. Springer, September 2004.

[17] Hongseok Yang. *Local Reasoning for Stateful Programs*. PhD thesis, Graduate College of the University of Illinois at Urbana-Champaign, 2001.