Verifying the Heap

Original Research: [Rey02]

Presenter: Alexander Weinert
Email: alexander.weinert@rwth-aachen.de

Apr 30, 2014

Motivation

Up until now: Each variable holds a single value from $\mathbb{N}$

Most prominent missing feature: References

Motivation

Up until now: Each variable holds a single value from $\mathbb{N}$

Most prominent missing feature: References

Excludes lots of interesting concepts: Lists, Trees, Graphs, OOP

Solution: Introduce formal handling of heap

Outline

Notation

Partial functions	$f : A \rightharpoonup B$
Undefined point	$f(x) = \perp$
Evaluation	$[\![c]\!]_s$ $[\![e]\!]_s$
Arbitrary Value	$f : x \mapsto$ -

Using the Heap

Four primitives:

- Allocation
- Lookup
- Mutation
- Deallocation

Using the Heap

Four primitives:

Allocation	$x := \textbf{alloc}(e_1, \ldots, e_n)$
Lookup	
Mutation	
Deallocation	

Using the Heap

Four primitives:

Allocation	$x := \textbf{alloc}(e_1, \ldots, e_n)$
Lookup	$x := [e]$
Mutation	
Deallocation	

Using the Heap

Four primitives:

Allocation	$x := \textbf{alloc}(e_1, \ldots, e_n)$
Lookup	$x := [e]$
Mutation	$[e_1] := e_2$
Deallocation	

Using the Heap

Four primitives:

Allocation	$x := \textbf{alloc}(e_1, \ldots, e_n)$
Lookup	$x := [e]$
Mutation	$[e_1] := e_2$
Deallocation	$\textbf{free}(e)$

New Language Obj

$\mathit{comm} :=$

New Language OBJ

$$\begin{array}{rcll} comm := & & \textbf{skip} \quad \mid \quad x := expr_a & \\ & \mid & comm;comm & \\ & \mid & \textbf{if}\ expr_b\ \textbf{then}\ comm\ \textbf{then}\ comm & \text{IMP} \\ & \mid & \textbf{while}\ expr_b\ \textbf{do}\ comm & \end{array}$$

$$\begin{array}{rlll} expr_a := & expr_a + expr_a & \mid \quad \ldots & \text{IMP} \\ expr_b := & expr_b \wedge expr_b & \mid \quad \ldots & \end{array}$$

New Language Obj

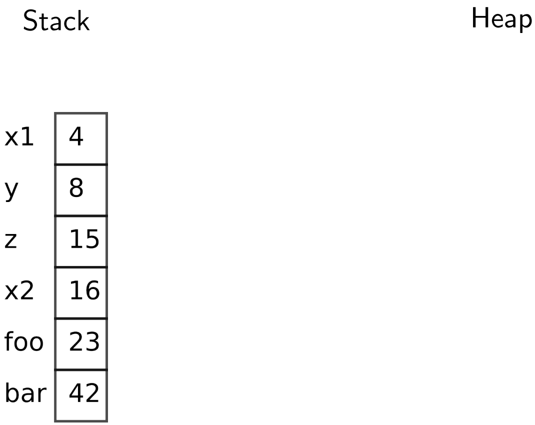
$$
\begin{array}{rcll}
\mathit{comm} := & & \textbf{skip} \quad | \quad x := \mathit{expr}_a & \\
& | & \mathit{comm};\mathit{comm} & \\
& | & \textbf{if } \mathit{expr}_b \textbf{ then } \mathit{comm} \textbf{ then } \mathit{comm} & \textsc{Imp} \\
& | & \textbf{while } \mathit{expr}_b \textbf{ do } \mathit{comm} & \\
& & & \\
& | & x := \textbf{alloc}(\mathit{expr}_a, \ldots, \mathit{expr}_a) & \\
& | & x := [\mathit{expr}_a] \quad | \quad [\mathit{expr}_a] := \mathit{expr}_a & \textsc{Obj} \\
& | & \textbf{free}(\mathit{expr}_a) & \\
& & & \\
\mathit{expr}_a := & & \mathit{expr}_a + \mathit{expr}_a \quad | \quad \ldots & \\
\mathit{expr}_b := & & \mathit{expr}_b \wedge \mathit{expr}_b \quad | \quad \ldots & \textsc{Imp}
\end{array}
$$

Introducing the Heap (Intuition)

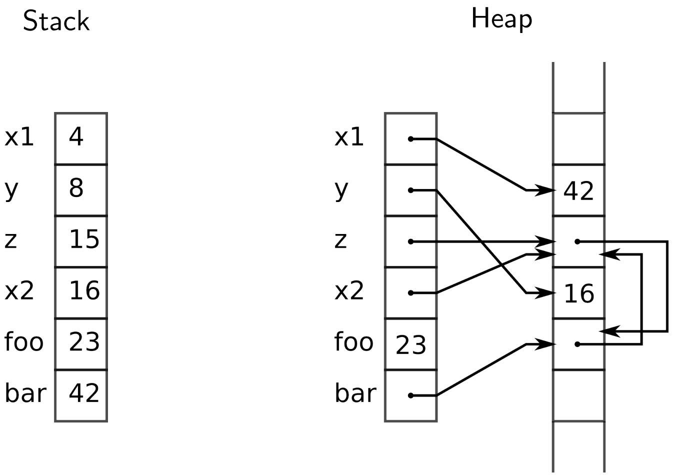
Stack

Heap

Introducing the Heap (Intuition)



Introducing the Heap (Intuition)



Observations

- Still a fixed (finite) set of variables: *Var*
- Variables can hold *Atom*s or *Add*resses
- *Add*resses point to either *Atom*s or *Add*resses

Observations

- Still a fixed (finite) set of variables: *Var*
- Variables can hold *Atom*s or *Add*resses
- *Add*resses point to either *Atom*s or *Add*resses

Idea: Function from variables to heap,
Function from heap to values

Introducing the Heap (formally)

Set of variables	Var	
Set of atoms	Atom	
Set of addresses	Add	
Set of values	$\mathit{Atom} \cup \mathit{Add}$	$=: \mathit{Val}$
	$\mathit{Atom} \cap \mathit{Add}$	$\stackrel{!}{=} \varnothing$

Introducing the Heap (formally)

Set of variables	Var	
Set of atoms	$Atom$	
Set of addresses	Add	
Set of values	$Atom \cup Add$	$=: Val$
	$Atom \cap Add$	$\stackrel{!}{=} \emptyset$

Stack	$s :$	$Var \rightharpoonup Val$
Heap	$h :$	$Add \rightharpoonup Val$

Introducing the Heap (formally)

Set of variables	Var
Set of atoms	Atom
Set of addresses	Add
Set of values	$\mathit{Atom} \cup \mathit{Add} \quad =: \mathit{Val}$
	$\mathit{Atom} \cap \mathit{Add} \quad \overset{!}{=} \varnothing$

Stack	$s: \quad \mathit{Var} \rightharpoonup \mathit{Val}$
Heap	$h: \quad \mathit{Add} \rightharpoonup \mathit{Val}$

New configuration: $\langle s, h \rangle$

Introducing the Heap (formally)

Set of variables	Var	
Set of atoms	$Atom$	
Set of addresses	Add	
Set of values	$Atom \cup Add$	$=: Val$
	$Atom \cap Add$	$\stackrel{!}{=} \varnothing$

Stack	$s :$	$Var \rightharpoonup Val$
Heap	$h :$	$Add \rightharpoonup Val$

New configuration: $\langle s, h \rangle$

New execution relation: $(\langle s, h \rangle, c) \Downarrow \langle s', h' \rangle$

Address arithmetic

Two more constraints:

- Atoms should still be integers
- Should model address arithmetic

Address arithmetic

Two more constraints:

- Atoms should still be integers
- Should model address arithmetic

$$\Rightarrow \mathit{Atom} \subsetneq \mathbb{N}, \mathit{Add} \subsetneq \mathbb{N}$$

Inference Rules

$$\textsf{Imp}\ \frac{c \in \textrm{IMP}}{(\langle s, h\rangle, c) \Downarrow \langle s', h\rangle}$$

Inference Rules

$$\textsf{Imp}\frac{c \in \textrm{I}\small{\textrm{MP}} \qquad (s, c) \Downarrow s'}{(\langle s, h\rangle, c) \Downarrow \langle s', h\rangle}$$

Inference Rules

$$\text{Imp}\,\frac{c \in \textsc{Imp} \qquad (s, c) \Downarrow s'}{(\langle s, h \rangle, c) \Downarrow \langle s', h \rangle}$$

$$\text{lookup}\,\frac{}{(\langle s, h \rangle, x := [e]) \Downarrow \langle s[x/v], h \rangle}$$

Inference Rules

$$\textsf{Imp}\,\frac{c \in \textsc{Imp} \qquad (s, c) \Downarrow s'}{(\langle s, h\rangle, c) \Downarrow \langle s', h\rangle}$$

$$\textsf{lookup}\,\frac{[\![e]\!]_s = a \qquad\qquad\qquad\qquad a \in Add}{(\langle s, h\rangle, x := [e]) \Downarrow \langle s[x/v], h\rangle}$$

Inference Rules

$$\text{Imp}\,\frac{c \in \textsc{Imp} \qquad (s, c) \Downarrow s'}{(\langle s, h\rangle, c) \Downarrow \langle s', h\rangle}$$

$$\text{lookup}\,\frac{[\![e]\!]_s = a \qquad h(a) = v \neq \bot \qquad a \in Add}{(\langle s, h\rangle, x := [e]) \Downarrow \langle s[x/v], h\rangle}$$

Inference Rules

$$\text{Imp}\frac{c \in \textsc{Imp} \qquad (s, c) \Downarrow s'}{(\langle s, h\rangle, c) \Downarrow \langle s', h\rangle}$$

$$\text{lookup}\frac{[\![e]\!]_s = a \qquad h(a) = v \neq \bot \qquad a \in \mathit{Add}}{(\langle s, h\rangle, x := [e]) \Downarrow \langle s[x/v], h\rangle}$$

$$\text{update}\frac{}{(\langle s, h\rangle, [e_1] := e_2) \Downarrow \langle s, h[a/v]\rangle}$$

Inference Rules

$$\textsf{Imp}\ \frac{c \in \textsc{Imp} \qquad (s, c) \Downarrow s'}{(\langle s, h\rangle, c) \Downarrow \langle s', h\rangle}$$

$$\textsf{lookup}\ \frac{[\![e]\!]_s = a \qquad h(a) = v \neq \bot \qquad a \in Add}{(\langle s, h\rangle, x := [e]) \Downarrow \langle s[x/v], h\rangle}$$

$$\textsf{update}\ \frac{[\![e_1]\!]_s = a \qquad\qquad\qquad a \in Add}{(\langle s, h\rangle, [e_1] := e_2) \Downarrow \langle s, h[a/v]\rangle}$$

Inference Rules

$$\text{Imp}\;\frac{c \in \textsc{Imp} \qquad (s, c) \Downarrow s'}{(\langle s, h\rangle, c) \Downarrow \langle s', h\rangle}$$

$$\text{lookup}\;\frac{[\![e]\!]_s = a \qquad h(a) = v \neq \bot \qquad a \in Add}{(\langle s, h\rangle, x := [e]) \Downarrow \langle s[x/v], h\rangle}$$

$$\text{update}\;\frac{[\![e_1]\!]_s = a \qquad [\![e_2]\!]_s = v \qquad a \in Add}{(\langle s, h\rangle, [e_1] := e_2) \Downarrow \langle s, h[a/v]\rangle}$$

Inference Rules(cont.)

$$\text{free}\ \frac{}{(\langle s, h\rangle, \textbf{free}(e)) \Downarrow \langle s, h[\quad\quad]\rangle}$$

Inference Rules(cont.)

$$\text{free}\,\frac{[\![e]\!]_s = a \qquad a \in Add}{(\langle s, h\rangle, \mathbf{free}(e)) \Downarrow \langle s, h[\qquad]\rangle}$$

Inference Rules(cont.)

$$\text{free}\ \frac{[\![e]\!]_s = a \qquad a \in Add}{(\langle s, h\rangle, \textbf{free}(e)) \Downarrow \langle s, h[a/\bot]\rangle}$$

Inference Rules(cont.)

$$\text{free}\ \frac{[[e]]_s = a \qquad a \in Add}{(\langle s, h\rangle, \mathbf{free}(e)) \Downarrow \langle s, h[a/\bot]\rangle}$$

$$\text{alloc}\ \frac{}{(\langle s, h\rangle, x := \mathbf{alloc}(e_1, \ldots, e_n)) \Downarrow \langle s', h'\rangle}$$

Where and

Inference Rules(cont.)

$$\text{free}\;\frac{[[e]]_s = a \qquad a \in Add}{(\langle s, h\rangle, \textbf{free}(e)) \Downarrow \langle s, h[a/\bot]\rangle}$$

$$\text{alloc}\;\frac{a \in Add \qquad h(a + i \text{ - } 1) = \bot}{(\langle s, h\rangle, x := \textbf{alloc}(e_1, \ldots, e_n)) \Downarrow \langle s', h'\rangle}$$

Where and

Inference Rules(cont.)

$$\text{free}\ \frac{[[e]]_s = a \qquad a \in Add}{(\langle s, h\rangle, \mathbf{free}(e)) \Downarrow \langle s, h[a/\bot]\rangle}$$

$$\text{alloc}\ \frac{a \in Add \qquad h(a + i \text{ - } 1) = \bot \qquad [[e_i]]_s = v_i}{(\langle s, h\rangle, x := \mathbf{alloc}(e_1, \ldots, e_n)) \Downarrow \langle s', h'\rangle}$$

Where and

Inference Rules(cont.)

$$\text{free}\ \frac{[\![e]\!]_s = a \qquad a \in Add}{(\langle s, h\rangle, \textbf{free}(e)) \Downarrow \langle s, h[a/\bot]\rangle}$$

$$\text{alloc}\ \frac{a \in Add \qquad h(a + i - 1) = \bot \qquad [\![e_i]\!]_s = v_i}{(\langle s, h\rangle, x := \textbf{alloc}(e_1, \ldots, e_n)) \Downarrow \langle s', h'\rangle}$$

Where $\quad s' = s[x/a] \quad$ and

Inference Rules(cont.)

$$\text{free}\;\frac{[\![e]\!]_s = a \qquad a \in Add}{(\langle s, h\rangle, \textbf{free}(e)) \Downarrow \langle s, h[a/\bot]\rangle}$$

$$\text{alloc}\;\frac{a \in Add \qquad h(a + i - 1) = \bot \qquad [\![e_i]\!]_s = v_i}{(\langle s, h\rangle, x := \textbf{alloc}(e_1, \ldots, e_n)) \Downarrow \langle s', h'\rangle}$$

Where $\quad s' = s[x/a] \quad$ and $\quad h' = h[(a+0)/v_1]\ldots[(a+n-1)/v_n]$

More Motivation

We have: Operational semantics of OBJ

More Motivation

We have: Operational semantics of OBJ

We want: Verification of OBJ programs

More Motivation

We have: Operational semantics of OBJ

We want: Verification of OBJ programs

Solution: Extend axiomatic semantics of of IMP

Reminder Hoare Calculus [Hoa69]

Judgments of the form $\{Pre\}\, c\, \{Post\}$

Reminder Hoare Calculus [Hoa69]

Judgments of the form $\{Pre\}\, c\, \{Post\}$

c:	Program in IMP
Pre, $Post$:	Logical formulas over $\mathbb{N}$, Var
	depends on underlying theory

Reminder Hoare Calculus [Hoa69]

Judgments of the form $\{Pre\}\, c\, \{Post\}$

c:	Program in IMP
Pre, $Post$:	Logical formulas over $\mathbb{N}$, Var
	depends on underlying theory

New underlying theory: Separation Logic

Separation Logic

Four new operators:

Empty Heap	**emp**
Singleton Heap	$\cdot \mapsto \cdot$
Separating Conjunction	$\cdot * \cdot$
Separating Implication	$\cdot \mathrel{-\!\!*} \cdot$

Inference Rules [Rey02]

All rules of Hoare Calculus remain sound

Inference Rules [Rey02]

All rules of Hoare Calculus remain sound

$$\text{update}\;\frac{}{\{e_1 \mapsto -\}\,[e_1] = e_2\,\{e_1 \mapsto e_2\}}$$

Inference Rules [Rey02]

All rules of Hoare Calculus remain sound

$$\text{update}\ \frac{}{\{e_1 \mapsto \text{-}\}\,[e_1] = e_2\,\{e_1 \mapsto e_2\}}$$

$$\text{lookup}\ \frac{}{\{e \mapsto v \wedge x = \text{-}\}\,x := [e]\,\{e \mapsto v \wedge x = v\}}$$

Inference Rules (cont.) [Rey02]

$$\text{free}\ \frac{}{\{e \mapsto \text{-}\}\ \mathbf{free}(e)\ \{\mathbf{emp}\}}$$

Inference Rules (cont.) [Rey02]

$$\text{free}\ \frac{}{\{e \mapsto \text{-}\}\ \mathbf{free}(e)\ \{\mathbf{emp}\}}$$

$$\text{alloc}\ \frac{}{\{\mathbf{emp}\}\ x := \mathbf{alloc}(e_1, \ldots, e_n)\ \{x \mapsto e_1 * \ldots * (x + n \text{ - } 1) \mapsto e_n\}}$$

Example

$$\text{update}\ \frac{}{\{e_1 \mapsto -\}\ [e_1] = e_2\ \{e_1 \mapsto e_2\}} \qquad \text{free}\ \frac{}{\{e \mapsto -\}\ \mathbf{free}(e)\ \{\mathbf{emp}\}}$$

$$\text{lookup}\ \frac{}{\{e \mapsto v\}\ x := [e]\ \{e \mapsto v \wedge x = v\}}$$

$$\text{alloc}\ \frac{}{\{\mathbf{emp}\}\ x := \mathbf{alloc}(e_1, \ldots, e_n)\ \{x \mapsto e_1 * \ldots * (x + n - 1) \mapsto e_n\}}$$

Example

$$\text{update}\ \frac{}{\{e_1 \mapsto -\}\ [e_1] = e_2\ \{e_1 \mapsto e_2\}} \qquad \text{free}\ \frac{}{\{e \mapsto -\}\ \textbf{free}(e)\ \{\textbf{emp}\}}$$

$$\text{lookup}\ \frac{}{\{e \mapsto v\}\ x := [e]\ \{e \mapsto v \wedge x = v\}}$$

$$\text{alloc}\ \frac{}{\{\textbf{emp}\}\ x := \textbf{alloc}(e_1, \ldots, e_n)\ \{x \mapsto e_1 * \ldots * (x + n - 1) \mapsto e_n\}}$$

To show:

$$\frac{}{\{x \mapsto 23 * y \mapsto 15\}\ \mathit{free}(x)\ \{y \mapsto 15\}}$$

Example

$$\text{update}\ \frac{}{\{e_1 \mapsto \text{-}\}\ [e_1] = e_2\ \{e_1 \mapsto e_2\}} \qquad \text{free}\ \frac{}{\{e \mapsto \text{-}\}\ \mathbf{free}(e)\ \{\mathbf{emp}\}}$$

$$\text{lookup}\ \frac{}{\{e \mapsto v\}\ x := [e]\ \{e \mapsto v \wedge x = v\}}$$

$$\text{alloc}\ \frac{}{\{\mathbf{emp}\}\ x := \mathbf{alloc}(e_1, \ldots, e_n)\ \{x \mapsto e_1 * \ldots * (x + n \text{ - } 1) \mapsto e_n\}}$$

To show:

$$\frac{\ldots}{\{x \mapsto 23 * y \mapsto 15\}\ \mathit{free}(x)\ \{y \mapsto 15\}}$$

Framing rule [Rey02]

$$\text{frame}\,\frac{\{p\}\; c\; \{q\}}{\{p * r\}\; c\; \{q * r\}}\,,$$

where c does not modify variables occurring in r

Framing rule [Rey02]

$$\text{frame}\,\frac{\{p\}\;c\;\{q\}}{\{p * r\}\;c\;\{q * r\}},$$

where c does not modify variables occurring in r

"Consequence rule of Separation Logic"

Example, take 2

$$\text{cons}\,\frac{}{\{x \mapsto 23 * y \mapsto 15\}\ \mathit{free}(x)\ \{y \mapsto 15\}}$$

Example, take 2

$$\text{cons}\ \dfrac{\text{frame}\ \dfrac{}{\{x \mapsto 23 * y \mapsto 15\}\ \mathit{free}(x)\ \{\mathbf{emp} * y \mapsto 15\}}}{\{x \mapsto 23 * y \mapsto 15\}\ \mathit{free}(x)\ \{y \mapsto 15\}}$$

Example, take 2

$$\dfrac{\dfrac{\dfrac{}{\{x \mapsto 23\}\ \mathit{free}(x)\ \{\mathbf{emp}\}}\ \text{free}}{\{x \mapsto 23 * y \mapsto 15\}\ \mathit{free}(x)\ \{\mathbf{emp} * y \mapsto 15\}}\ \text{frame}}{\{x \mapsto 23 * y \mapsto 15\}\ \mathit{free}(x)\ \{y \mapsto 15\}}\ \text{cons}$$

What now?

What now?

- Verify programs using the heap
  - e.g. Garbage Collector [Yan01]
- Shape analysis [DOY06]
- Prove information hiding of library [OYR04]

Information hiding [OYR04]

Previously: Program as single, monolithic procedure

Information hiding [OYR04]

Previously: Program as single, monolithic procedure

Now: Program separated into several functions, libraries

Libraries in Separation Logic

Procedure Names:	$k_1, \ldots, k_n$
Interface Specification:	$\{P_1\}k_1\{Q_1\}[X_1], \ldots, \{P_n\}k_n\{Q_n\}[X_n]$
Implementations:	$c_1, \ldots, c_n$
Resource Invariant	r
Internal Variables:	Y

Using Libraries in Separation Logic

We have to show:

$$\{P_i * r\}c_i\{Q_i * r\}, \text{ for each procedure } k_i$$

Using Libraries in Separation Logic

We have to show:

$$\{P_i * r\}c_i\{Q_i * r\}, \text{ for each procedure } k_i$$

Then we can show:

$$\{P\}c\{Q\}, \text{ for the main program}$$

using the assumptions:

$$\{P_i\}c_i\{Q_i\}, \text{ for all library procedures,}$$

if

$$c \text{ does not use variables in } r$$

Benefits

- Change of implementation: Just prove new implementation
- Wrong use of interface variables can be precluded

Downsides

- Definition of modules is very clunky
- Use of interface variables: Either full access or none

Summary

- General idea: Introduce mathematical handling of heap

Summary

- General idea: Introduce mathematical handling of heap
- Formally defined the heap

Summary

- General idea: Introduce mathematical handling of heap
- Formally defined the heap
  - Syntactic definitions
  - Operational Semantics
  - Axiomatic Semantics (Hoare Calculus + Separation Logic)

Summary

- General idea: Introduce mathematical handling of heap
- Formally defined the heap
  - Syntactic definitions
  - Operational Semantics
  - Axiomatic Semantics (Hoare Calculus + Separation Logic)
- Example for usage of Axiomatic Semantics

Model Checking [Clarke, Emerson, Sifakis]

Idea: Explore state space

Model Checking [Clarke, Emerson, Sifakis]

Idea: Explore state space

May be large, but finite for stack programs

Model Checking [Clarke, Emerson, Sifakis]

Idea: Explore state space

May be large, but finite for stack programs

Problem: Infinitely many states for heap programs

Model Checking [Clarke, Emerson, Sifakis]

Idea: Explore state space

May be large, but finite for stack programs

Problem: Infinitely many states for heap programs

Graph Grammars [HNR10]

	String Grammars	Graph Grammars
Description of:	Set of strings	Set of graphs
Atoms:	Characters	Objects of the heap
Derivation:	Replace Nonterminals	Replace inactive parts of heap

⇒ Finite description of all possible heap configurations

Graph Grammars [HNR10]

	String Grammars	Graph Grammars
Description of:	Set of strings	Set of graphs
Atoms:	Characters	Objects of the heap
Derivation:	Replace Nonterminals	Replace inactive parts of heap

⇒ Finite description of all possible heap configurations

⇒ Finite state space

Dino Distefano, Peter W. O'Hearn, and Hongseok Yang.
A local shape analysis based on separation logic.
In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 287–302. Springer, 2006.

Jonathan Heinen, Thomas Noll, and Stefan Rieger.
Juggrnaut: Graph grammar abstraction for unbounded heap structures.
Electronic Notes in Theoretical Computer Science, 266:93–107, 2010.

Charles A. R. Hoare.
An axiomatic basis for computer programming.
Communications of the ACM, 12(10):576–580, 1969.

Peter W. O'Hearn, Hongseok Yang, and John C. Reynolds.
Separation and information hiding.
In *ACM SIGPLAN Notices*, volume 39, pages 268–280. ACM, 2004.

John C. Reynolds.

Separation logic: A logic for shared mutable data structures.
In *Logic in Computer Science, 2002. Proceedings. 17th Annual IEEE Symposium on*, pages 55–74. IEEE, 2002.

Hongseok Yang.
An example of local reasoning in bi pointer logic: the schorr-waite graph marking algorithm.
SPACE, 1, 2001.