Lectures on Reasoning about Shared Mutable Data Structure

Part 1

John C. Reynolds

Department of Computer Science
Carnegie Mellon University
and
BRICS, Department of Computer Science
University of Aarhus

Joint work with
Peter W. O'Hearn and Hongseok Yang

June 18, 2001, corrected June 29

The topic of these lectures is an extension of Hoare logic [1, 2] to the specification and proof of programs in which data structures can contain several pointers to the same location, whose contents can be modified by the program.

The central idea, of an "independent conjunction" $p * q$ that asserts that p and q hold for separate parts of the data structure, is implicit in early work by Burstall [3]. It was explicitly described by Reynolds in lectures in the fall of 1999; then an intuitionistic logic based on this idea was discovered independently by Reynolds [20] and by Ishtiaq and O'Hearn [21]. Ishtiaq and O'Hearn realized that this logic was a special case of a bunched implication logic [18], and also introduced an "independent implication" operation.

In addition, Ishtiaq and O'Hearn devised a classical version of the logic that is more expressive than the intuitionistic version. (In particular, it can express storage deallocation.) Recently, Reynolds extended the classical version by adding pointer arithmetic [23]; it is this extension that will be used in these lectures. We will also make use of O'Hearn's "frame rule" [26, 27], which permits local reasoning about components of programs.

Other recent relevant papers include work by Calcagno on the treatment of garbage collection in the intuitionistic logic [24], by Yang on a proof of the Schorr-Waite Algorithm [25], and by Walker and Morrisett on a type system with goals that are similar to our logics [22].

The research of my own that is described in these lectures was supported by National Science Foundation Grant CCR–9804014.

References

[1] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580 and 583, October 1969.

[2] C. A. R. Hoare. Proof of a program: FIND. *Communications of the ACM*, 14(1):39–45, January 1971.

[3] Rodney M. Burstall. Some techniques for proving correctness of programs which alter data structures. In Bernard Meltzer and Donald Michie, editors, *Machine Intelligence 7*, pages 23–50. Edinburgh University Press, Edinburgh, Scotland, 1972.

[4] T. Kowaltowski. Correctness of programs manipulating data structures. Memorandum ERL-M404, University of California, Berkeley, California, September 1973.

[5] Stephen A. Cook and Derek C. Oppen. An assertion language for data structures. In *Conference Record of the Second ACM Symposium on Principles of Programming Languages*, pages 160–166, New York, 1975. ACM.

[6] Derek C. Oppen and Stephen A. Cook. Proving assertions about programs that manipulate data structures. In *Proceedings of Seventh Annual ACM Symposium on Theory of Computing*, pages 107–116, New York, 1975. ACM.

[7] Joseph M. Morris. A general axiom of assignment; assignment and linked data structures; a proof of the Schorr-Waite algorithm. In Manfried Broy and Gunther Schmidt,

editors, *Theoretical Foundations of Programming Methodology*, pages 25–51. D. Reidel, Dordrecht, Holland, 1982.

[8] Ian A. Mason. *The Semantics of Destructive Lisp*. CSLI Lecture Notes Number 5. Center for the Study of Language and Information, Menlo Park, CA, 1986.

[9] Ian A. Mason. Verification of programs that destructively manipulate data. *Science of Computer Programming*, 10(2):177–210, April 1988.

[10] Ian A. Mason and Carolyn Talcott. Equivalence in functional languages with effects. *Journal of Functional Programming*, 1(3):287–327, July 1991.

[11] Furio Honsell, Ian A. Mason, Scott Smith, and Carolyn Talcott. A variable typed logic of effects. *Information and Computation*, 119(1):55–90, May 15, 1995.

[12] Ian A. Mason. A first order logic of effects. *Theoretical Computer Science*, 185(2):277–318, October 20, 1997.

[13] Ian D. B. Stark. *Names and Higher-Order Functions*. Ph. D. dissertation, University of Cambridge, Cambridge, England, December 1994.

[14] Ian D. B. Stark. Categorical models for local names. *Lisp and Symbolic Computation*, 9(1):77–107, February 1996.

[15] Ian D. B. Stark. Names, equations, relations: Practical ways to reason about *new*. *Fundamenta Informaticae*, 33:369–396, 1998.

[16] Andrew M. Pitts and Ian D. B. Stark. Operational reasoning for functions with local state. In Andrew D. Gordon and Andrew M. Pitts, editors, *Higher Order Operational Techniques in Semantics*, pages 227–273. Cambridge University Press, 1998.

[17] John C. Reynolds. *Theories of Programming Languages*. Cambridge University Press, Cambridge, England, 1998.

[18] Peter W. O'Hearn and David J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, June 1999.

[19] Cristiano Calcagno, Samin Ishtiaq, and Peter W. O'Hearn. Semantic analysis of pointer aliasing, allocation and disposal in Hoare logic. unpublished, 2000.

[20] John C. Reynolds. Intuitionistic reasoning about shared mutable data structure. In Jim Davies, Bill Roscoe, and Jim Woodcock, editors, *Millennial Perspectives in Computer Science*, pages 303–321, Houndsmill, Hampshire, 2000. Palgrave.

[21] Samin Ishtiaq and Peter W. O'Hearn. BI as an assertion language for mutable data structures. In *Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 14–26, New York, 2001. ACM.

[22] David Walker and Greg Morrisett. Alias types for recursive data structures. In *Preliminary Proceedings of the Third Workshop on Types in Compilation (TIC 2000)*. School of

Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, 2000. Report CMU–CS–00–161.

[23] John C. Reynolds and Peter W. O'Hearn. Reasoning about shared mutable data structure (abstract of invited lecture). In Fritz Henglein, John Hughes, Henning Makholm, and Henning Niss, editors, *SPACE 2001: Informal Proceedings of Workshop on Semantics, Program Analysis and Computing Environments for Memory Management*, page 7. IT University of Copenhagen, 2001. The slides for this lecture are available at ftp://ftp.cs.cmu.edu/user/jcr/spacetalk.ps.gz.

[24] Cristiano Calcagno. Program logics in the presence of garbage collection (abstract). In Fritz Henglein, John Hughes, Henning Makholm, and Henning Niss, editors, *SPACE 2001: Informal Proceedings of Workshop on Semantics, Program Analysis and Computing Environments for Memory Management*, page 11. IT University of Copenhagen, 2001.

[25] Hongseok Yang. An example of local reasoning in BI pointer logic: The Schorr-Waite graph marking algorithm. In Fritz Henglein, John Hughes, Henning Makholm, and Henning Niss, editors, *SPACE 2001: Informal Proceedings of Workshop on Semantics, Program Analysis and Computing Environments for Memory Management*, pages 41–68. IT University of Copenhagen, 2001.

[26] Peter W. O'Hearn, John C. Reynolds, and Hongseok Yang. Local reasoning about programs that alter data structures.

In *Proceedings of 15th Annual Conference of the European Association for Computer Science Logic: CSL 2001*, Lecture Notes in Computer Science, Berlin, 2001. Springer-Verlag. To appear.

[27] Hongseok Yang. *Local Reasoning for Stateful Programs*. Ph. D. dissertation, University of Illinois, Urbana-Champaign, Illinois, 2001. (expected).

Abstract Syntax of the Programming Language

$\langle \text{exp} \rangle ::= 0 \mid 1 \mid 2 \mid \cdots$
$\quad \mid \langle \text{var} \rangle \mid -\langle \text{exp} \rangle \mid \langle \text{exp} \rangle + \langle \text{exp} \rangle \mid \langle \text{exp} \rangle - \langle \text{exp} \rangle$
$\quad \mid \langle \text{exp} \rangle \times \langle \text{exp} \rangle \mid \cdots$

$\langle \text{boolexp} \rangle ::= \textbf{true} \mid \textbf{false}$
$\quad \mid \langle \text{exp} \rangle = \langle \text{exp} \rangle \mid \langle \text{exp} \rangle \neq \langle \text{exp} \rangle \mid \cdots$
$\quad \mid \neg \langle \text{boolexp} \rangle \mid \langle \text{boolexp} \rangle \wedge \langle \text{boolexp} \rangle \mid \langle \text{boolexp} \rangle \vee \langle \text{boolexp} \rangle$
$\quad \mid \langle \text{boolexp} \rangle \Rightarrow \langle \text{boolexp} \rangle \mid \langle \text{boolexp} \rangle \Leftrightarrow \langle \text{boolexp} \rangle$

$\langle \text{comm} \rangle ::= \langle \text{var} \rangle := \langle \text{exp} \rangle \mid \textbf{skip} \mid \langle \text{comm} \rangle \,;\, \langle \text{comm} \rangle$
$\quad \mid \textbf{if}\ \langle \text{boolexp} \rangle\ \textbf{then}\ \langle \text{comm} \rangle\ \textbf{else}\ \langle \text{comm} \rangle$
$\quad \mid \textbf{while}\ \langle \text{boolexp} \rangle\ \textbf{do}\ \langle \text{comm} \rangle$
$\quad \mid \textbf{newvar}\ \langle \text{var} \rangle\ \textbf{in}\ \langle \text{comm} \rangle$
$\quad \mid \langle \text{var} \rangle := \textbf{cons}(\langle \text{exp} \rangle, \ldots, \langle \text{exp} \rangle)$
$\quad \mid \langle \text{var} \rangle := [\langle \text{exp} \rangle]$
$\quad \mid [\langle \text{exp} \rangle] := \langle \text{exp} \rangle$
$\quad \mid \textbf{dispose}\ \langle \text{exp} \rangle$

Stores and Heaps

$$\begin{aligned}
\text{Atoms} &\subseteq \mathbf{Z} \\
\text{Locations} &\subseteq \mathbf{Z} \\
\text{Atoms} \cap \text{Locations} &= \{\} \\
\mathbf{nil} &\in \text{Atoms} \\
\\
\text{States} &= \text{Stores} \times \text{Heaps} \\
\text{Stores} &= \langle \text{var} \rangle \to \mathbf{Z} \\
\text{Heaps} &= \bigcup_{\substack{\text{fin} \\ A \subseteq \text{Locations}}} (A \to \mathbf{Z})
\end{aligned}$$

Eventually, stores will map

ordinary variables	into	$\mathbf{Z}$
sequence variables	into	sequences
tree variables	into	trees
assertion variables	into	$\text{Heaps} \to \mathbf{B}$
	$\vdots$	

Semantic Equations

$$\begin{aligned}
[\![-]\!]_{\text{exp}} &\in \langle \text{exp} \rangle \to \text{Stores} \to \mathbf{Z} \\
[\![-]\!]_{\text{boolexp}} &\in \langle \text{boolexp} \rangle \to \text{Stores} \to \mathbf{B} \\
[\![-]\!]_{\text{assert}} &\in \langle \text{assert} \rangle \to \text{Stores} \to \text{Heaps} \to \mathbf{B}
\end{aligned}$$

The Semantics of Expressions

$$[\![-]\!]_{\text{exp}} \in \langle \text{exp} \rangle \to \text{Stores} \to \mathbf{Z}.$$

$$[\![0]\!]_{\text{exp}} s = 0 \qquad \text{(and similarly for 1, 2, ...)},$$

$$[\![v]\!]_{\text{exp}} s = sv,$$

$$[\![-e]\!]_{\text{exp}} s = -[\![e]\!]_{\text{exp}} s,$$

$$[\![e_0 + e_1]\!]_{\text{exp}} s = [\![e_0]\!]_{\text{exp}} s + [\![e_1]\!]_{\text{exp}} s$$

$$\text{(and similarly for } -, \times, \ldots).$$

$$[\![-]\!]_{\text{boolexp}} \in \langle \text{boolexp} \rangle \to \text{Stores} \to \mathbf{B}.$$

$$[\![\mathbf{true}]\!]_{\text{boolexp}} s = \mathbf{true},$$

$$[\![\mathbf{false}]\!]_{\text{boolexp}} s = \mathbf{false},$$

$$[\![e_0 = e_1]\!]_{\text{boolexp}} s = ([\![e_0]\!]_{\text{exp}} s = [\![e_1]\!]_{\text{exp}} s)$$

$$\text{(and similarly for } \neq, \ldots),$$

$$[\![\neg b]\!]_{\text{boolexp}} s = \neg [\![b]\!]_{\text{boolexp}} s,$$

$$[\![b_0 \wedge b_1]\!]_{\text{boolexp}} s = [\![b_0]\!]_{\text{boolexp}} s \wedge [\![b_1]\!]_{\text{boolexp}} s$$

$$\text{(and similarly for } \vee, \Rightarrow, \Leftrightarrow).$$

An Example

$$\begin{aligned}[\![\mathsf{x}+\mathsf{y}=\mathsf{x}]\!]_{\mathrm{boolexp}}s &= ([\![\mathsf{x}+\mathsf{y}]\!]_{\mathrm{exp}}s = [\![\mathsf{x}]\!]_{\mathrm{exp}}s)\\ &= ([\![\mathsf{x}]\!]_{\mathrm{exp}}s + [\![\mathsf{y}]\!]_{\mathrm{exp}}s = [\![\mathsf{x}]\!]_{\mathrm{exp}}s)\\ &= (s\,\mathsf{x} + s\,\mathsf{y} = s\,\mathsf{x})\\ &= (s\,\mathsf{y} = 0).\end{aligned}$$

Substitution

We write

$$p/v_1 \to e_1, \ldots, v_n \to e_n$$

to denote the phrase obtained from p by simulataneously substituting each expression e_i for the variable v_i, with renaming of bound variables to avoid capture.

Proposition 1 *Let δ abbreviate the substitution*

$$v_1 \to e_1, \ldots, v_n \to e_n,$$

and for any store s, let

$$\widehat{s} = [\, v_1\colon [\![e_1]\!]_{\mathrm{exp}}s \mid \ldots \mid v_n\colon [\![e_n]\!]_{\mathrm{exp}}s\,].$$

If e is an expression (or boolean expression) whose free variables belong to $\{v_1, \ldots, v_n\}$, then

$$[\![e/\delta]\!]_{\mathrm{exp}}s = [\![e]\!]_{\mathrm{exp}}\widehat{s}.$$

Structural Operational Semantics

An execution (of a command) is a sequence

$$\gamma_0 \rightsquigarrow \gamma_1 \rightsquigarrow \cdots$$

of *configurations* γ_i related by the *transition relation* $\rightsquigarrow$.

A configuration is either:

- *nonterminal*: A command-state pair.
- *terminal*: A state or **abort**.

For example:

$$\langle \mathsf{x} := \mathsf{x} + 1 \,;\, \mathsf{y} := \mathsf{y} + \mathsf{x} \,;\, \mathsf{x} := \mathsf{x} + 1, ([\,\mathsf{x}\colon 3 \mid \mathsf{y}\colon 7\,], [\,]) \rangle \rightsquigarrow$$
$$\langle \mathsf{y} := \mathsf{y} + \mathsf{x} \,;\, \mathsf{x} := \mathsf{x} + 1, ([\,\mathsf{x}\colon 4 \mid \mathsf{y}\colon 7\,], [\,]) \rangle \rightsquigarrow$$
$$\langle \mathsf{x} := \mathsf{x} + 1, ([\,\mathsf{x}\colon 4 \mid \mathsf{y}\colon 11\,], [\,]) \rangle \rightsquigarrow$$
$$([\,\mathsf{x}\colon 5 \mid \mathsf{y}\colon 11\,], [\,]).$$

$$\langle \mathsf{x} := \mathbf{cons}(1, 2) \,;\, \mathsf{y} := [\mathsf{x}] \,;\, [\mathsf{x} + 1] := 3, ([\,\mathsf{x}\colon 3 \mid \mathsf{y}\colon 4\,], [\,]) \rangle \rightsquigarrow$$
$$\langle \mathsf{y} := [\mathsf{x}] \,;\, [\mathsf{x} + 1] := 3, ([\,\mathsf{x}\colon 37 \mid \mathsf{y}\colon 4\,], [\,37\colon 1 \mid 38\colon 2\,]) \rangle \rightsquigarrow$$
$$\langle [\mathsf{x} + 1] := 3, ([\,\mathsf{x}\colon 37 \mid \mathsf{y}\colon 1\,], [\,37\colon 1 \mid 38\colon 2\,]) \rangle \rightsquigarrow$$
$$([\,\mathsf{x}\colon 37 \mid \mathsf{y}\colon 1\,], [\,37\colon 1 \mid 38\colon 3\,]).$$

$$\langle \mathsf{x} := \mathbf{cons}(1, 2) \,;\, \mathsf{y} := [\mathsf{x} + 2] \,;\, [\mathsf{x} + 1] := 3, ([\,\mathsf{x}\colon 3 \mid \mathsf{y}\colon 4\,], [\,]) \rangle \rightsquigarrow$$
$$\langle \mathsf{y} := [\mathsf{x} + 2] \,;\, [\mathsf{x} + 1] := 3, ([\,\mathsf{x}\colon 37 \mid \mathsf{y}\colon 4\,], [\,37\colon 1 \mid 38\colon 2\,]) \rangle \rightsquigarrow$$
$$\mathbf{abort}.$$

Structural Operational Semantics: Notation

We write:

- $\gamma \rightsquigarrow^* \gamma'$ iff there is a finite execution from γ to γ'.
- $\gamma \uparrow$ iff there is an infinite execution beginning with γ.

Proposition 2 *Our semantics is* total*:*

- *For every nonterminal configuration γ, there is at least one configuration γ' such that $\gamma \rightsquigarrow \gamma'$, i.e., executions do not get* stuck*.*
- *Every execution sequence can be extended to a sequence that is either infinite or ends with a terminal configuration.*

The transition relation is defined by inference rules called *transition rules.*

Transition Rules for the Simple Imperative Language

- Assignment

$$\frac{}{\langle v := e, (s, h)\rangle \rightsquigarrow ([\, s \mid v\colon [\![e]\!]_{\mathrm{exp}} s\,], h)}$$

- **skip**

$$\frac{}{\langle \mathbf{skip}, (s, h)\rangle \rightsquigarrow (s, h)}$$

- Sequencing

$$\frac{\langle c_0, (s, h)\rangle \rightsquigarrow (s', h')}{\langle c_0 \,;\, c_1, (s, h)\rangle \rightsquigarrow \langle c_1, (s', h')\rangle}$$

$$\frac{\langle c_0, (s, h)\rangle \rightsquigarrow \langle c_0', (s', h')\rangle}{\langle c_0 \,;\, c_1, (s, h)\rangle \rightsquigarrow \langle c_0' \,;\, c_1, (s', h')\rangle}$$

$$\frac{\langle c_0, (s, h)\rangle \rightsquigarrow \mathbf{abort}}{\langle c_0 \,;\, c_1, (s, h)\rangle \rightsquigarrow \mathbf{abort}}$$

For example:

$$\langle \mathsf{x} := \mathsf{x} + 1, ([\,\mathsf{x}\colon 3 \mid \mathsf{y}\colon 7\,], [\,])\rangle \rightsquigarrow ([\,\mathsf{x}\colon 4 \mid \mathsf{y}\colon 7\,], [\,])$$

$$\langle \mathsf{x} := \mathsf{x} + 1 \,;\, \mathsf{y} := \mathsf{y} + \mathsf{x}, ([\,\mathsf{x}\colon 3 \mid \mathsf{y}\colon 7\,], [\,])\rangle \rightsquigarrow \langle \mathsf{y} := \mathsf{y} + \mathsf{x}, ([\,\mathsf{x}\colon 4 \mid \mathsf{y}\colon 7\,], [\,])\rangle$$

$$\langle (\mathsf{x} := \mathsf{x} + 1 \,;\, \mathsf{y} := \mathsf{y} + \mathsf{x}) \,;\, \mathsf{x} := \mathsf{x} + 1, ([\,\mathsf{x}\colon 3 \mid \mathsf{y}\colon 7\,], [\,])\rangle \rightsquigarrow \langle \mathsf{y} := \mathsf{y} + \mathsf{x} \,;\, \mathsf{x} := \mathsf{x} + 1, ([\,\mathsf{x}\colon 4 \mid \mathsf{y}\colon 7\,], [\,])\rangle$$

Conditional, **while**, and **newvar** Commands

- Conditional

  When $[\![b]\!]_{\text{boolexp}} s = \mathbf{true}$:

  $$\frac{}{\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, (s,h)\rangle \rightsquigarrow \langle c_0, (s,h)\rangle}$$

  When $[\![b]\!]_{\text{boolexp}} s = \mathbf{false}$:

  $$\frac{}{\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, (s,h)\rangle \rightsquigarrow \langle c_1, (s,h)\rangle}$$

- **while**

  When $[\![b]\!]_{\text{boolexp}} s = \mathbf{true}$:

  $$\frac{}{\langle \mathbf{while}\ b\ \mathbf{do}\ c, (s,h)\rangle \rightsquigarrow \langle c\,;\,\mathbf{while}\ b\ \mathbf{do}\ c, (s,h)\rangle}$$

  When $[\![b]\!]_{\text{boolexp}} s = \mathbf{false}$:

  $$\frac{}{\langle \mathbf{while}\ b\ \mathbf{do}\ c, (s,h)\rangle \rightsquigarrow (s,h)}$$

- **newvar**

  $$\frac{}{\langle \mathbf{newvar}\ v\ \mathbf{in}\ c, (s,h)\rangle \rightsquigarrow \langle c\,;\, v := n, ([\, s \mid v\colon m\,], h)\rangle}$$

  where $m \in \mathbf{Z}$ and $n = s\,v$.

(See [17, Section 6.2, pp. 128–130] for a discussion of the limitations of this transition rule for **newvar**.)

Transition Rules for the New Commands

- Construction

$$\frac{}{\langle v := \mathbf{cons}(e_0, \ldots, e_{n-1}), (s, h)\rangle \rightsquigarrow ([\, s \mid v{:}\, \ell \,], [\, h \mid \ell{:}\, [\![e_0]\!]_{\mathrm{exp}} s \mid \ldots \mid \ell + n - 1{:}\, [\![e_{n-1}]\!]_{\mathrm{exp}} s \,])}$$

  where $\ell, \ldots, \ell + n - 1 \in \mathrm{Locations} - \mathrm{dom}\, h$.

- Lookup

  When $[\![e]\!]_{\mathrm{exp}} s \in \mathrm{dom}\, h$:

$$\frac{}{\langle v := [e], (s, h)\rangle \rightsquigarrow ([\, s \mid v{:}\, h([\![e]\!]_{\mathrm{exp}} s) \,], h)}$$

  When $[\![e]\!]_{\mathrm{exp}} s \notin \mathrm{dom}\, h$:

$$\frac{}{\langle v := [e], (s, h)\rangle \rightsquigarrow \mathbf{abort}}$$

More New Commands

- Mutation

  When $[\![e]\!]_{\mathrm{exp}} s \in \mathrm{dom}\, h$:

$$\frac{}{\langle [e] := e', (s, h)\rangle \rightsquigarrow (s, [\, h \mid [\![e]\!]_{\mathrm{exp}} s \colon [\![e']\!]_{\mathrm{exp}} s \,])}$$

  When $[\![e]\!]_{\mathrm{exp}} s \notin \mathrm{dom}\, h$:

$$\frac{}{\langle [e] := e', (s, h)\rangle \rightsquigarrow \mathbf{abort}}$$

- Disposal

  When $[\![e]\!]_{\mathrm{exp}} s \in \mathrm{dom}\, h$:

$$\frac{}{\langle \mathbf{dispose}\ e, (s, h)\rangle \rightsquigarrow (s, h \rceil (\mathrm{dom}\, h - \{[\![e]\!]_{\mathrm{exp}} s\}))}$$

  When $[\![e]\!]_{\mathrm{exp}} s \notin \mathrm{dom}\, h$:

$$\frac{}{\langle \mathbf{dispose}\ e, (s, h)\rangle \rightsquigarrow \mathbf{abort}}$$

Free and Free Assigned Variables

$$\mathrm{FV}(c) = \text{“The variables occurring free in } c\text{”}$$
$$\mathrm{FA}(c) = \text{“The free variables assigned by } c\text{”}$$

$$\begin{aligned}
\mathrm{FA}(v := e) &= \{v\} \\
\mathrm{FA}(v := \mathbf{cons}(e_0, \ldots, e_{n-1})) &= \{v\} \\
\mathrm{FA}(v := [e]) &= \{v\} \\
\mathrm{FA}(\mathbf{skip}) &= \{\} \\
\mathrm{FA}([e] := e') &= \{\} \\
\mathrm{FA}(\mathbf{dispose}\ e) &= \{\} \\
\mathrm{FA}(c_0\ ;\ c_1) &= \mathrm{FA}(c_0) \cup \mathrm{FA}(c_1) \\
\mathrm{FA}(\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1) &= \mathrm{FA}(c_0) \cup \mathrm{FA}(c_1) \\
\mathrm{FA}(\mathbf{while}\ b\ \mathbf{do}\ c) &= \mathrm{FA}(c) \\
\mathrm{FA}(\mathbf{newvar}\ v\ \mathbf{in}\ c) &= \mathrm{FA}(c) - \{v\}
\end{aligned}$$

Proposition 3 *If* $\langle c, (s,h)\rangle \rightsquigarrow^* (s',h')$ *and* $v \notin \mathrm{FA}(c)$, *then* $s'v = s\,v$.

Substitution

Proposition 4 *Let c be a command such that $\mathrm{FV}(c) \subseteq \{v_1, \ldots, v_n\}$, let δ be a substitution*

$$v_1 \to e_1, \ldots, v_n \to e_n$$

such that, when $v_i \in \mathrm{FA}(c)$, e_i is a variable that does not occur in any other e_j. For any store s, let

$$\widehat{s} = [\, v_1 \colon [\![e_1]\!]_{\mathrm{exp}} s \mid \ldots \mid v_n \colon [\![e_n]\!]_{\mathrm{exp}} s \,].$$

Then

- *If $\langle c/\delta, (s,h)\rangle \rightsquigarrow (s', h')$, then $\langle c, (\widehat{s}, h)\rangle \rightsquigarrow (\widehat{s'}.h')$.*
- *If $\langle c/\delta, (s,h)\rangle \rightsquigarrow \langle \tilde{c}, (s', h')\rangle$, then there is a command c' such that $\langle c, (\widehat{s}, h)\rangle \rightsquigarrow \langle c', (\widehat{s'}.h')\rangle$ and $\tilde{c} = c'/\delta$.*
- *If $\langle c/\delta, (s,h)\rangle \rightsquigarrow$ **abort**, then $\langle c, (\widehat{s}, h)\rangle \rightsquigarrow$ **abort**.*

and therefore

- *If $\langle c/\delta, (s,h)\rangle \rightsquigarrow^* (s', h')$, then $\langle c, (\widehat{s}, h)\rangle \rightsquigarrow^* (\widehat{s'}.h')$.*
- *If $\langle c/\delta, (s,h)\rangle \rightsquigarrow^*$ **abort**, then $\langle c, (\widehat{s}, h)\rangle \rightsquigarrow^*$ **abort**.*
- *If $\langle c/\delta, (s,h)\rangle \uparrow$, then $\langle c, (\widehat{s}, h)\rangle \uparrow$.*

Splitting the Heap

$$h_0 \perp h_1 \text{ iff } \operatorname{dom} h_0 \cap \operatorname{dom} h_1 = \{\}$$

$$\operatorname{dom}(h_0 \cdot h_1) = \operatorname{dom} h_0 \cup \operatorname{dom} h_1$$

$$(h_0 \cdot h_1)\ell = \begin{cases} h_0\ell & \text{when } \ell \in \operatorname{dom} h_0 \\ h_1\ell & \text{when } \ell \in \operatorname{dom} h_1. \end{cases}$$

Proposition 5 *Suppose $h = h_0 \cdot h_1$, where $h_0 \perp h_1$. Then*

- *If $\langle c, (s, h)\rangle \rightsquigarrow$ **abort**, then $\langle c, (s, h_0)\rangle \rightsquigarrow$ **abort**.*
- *If $\langle c, (s, h)\rangle \rightsquigarrow \langle c', (s', h')\rangle$ then $\langle c, (s, h_0)\rangle \rightsquigarrow$ **abort** or $\langle c, (s, h_0)\rangle \rightsquigarrow \langle c', (s', h'_0)\rangle$, where $h'_0 \perp h_1$ and $h' = h'_0 \cdot h_1$.*
- *If $\langle c, (s, h)\rangle \rightsquigarrow (s', h')$ then $\langle c, (s, h_0)\rangle \rightsquigarrow$ **abort** or $\langle c, (s, h_0)\rangle \rightsquigarrow (s', h'_0)$, where $h'_0 \perp h_1$ and $h' = h'_0 \cdot h_1$.*

and therefore

- *If $\langle c, (s, h)\rangle \rightsquigarrow^*$ **abort**, then $\langle c, (s, h_0)\rangle \rightsquigarrow^*$ **abort**.*
- *If $\langle c, (s, h)\rangle \rightsquigarrow^* (s', h')$ then $\langle c, (s, h_0)\rangle \rightsquigarrow^*$ **abort** or $\langle c, (s, h_0)\rangle \rightsquigarrow^* (s', h'_0)$, where $h'_0 \perp h_1$ and $h' = h'_0 \cdot h_1$.*
- *If $\langle c, (s, h)\rangle \uparrow$ then $\langle c, (s, h_0)\rangle \rightsquigarrow^*$ **abort** or $\langle c, (s, h_0)\rangle \uparrow$.*

Syntax of Assertions

$$
\begin{aligned}
\langle \text{assert} \rangle ::= {} & \langle \text{boolexp} \rangle \\
& \mid \neg \langle \text{assert} \rangle \mid \langle \text{assert} \rangle \wedge \langle \text{assert} \rangle \mid \langle \text{assert} \rangle \vee \langle \text{assert} \rangle \\
& \mid \langle \text{assert} \rangle \Rightarrow \langle \text{assert} \rangle \mid \langle \text{assert} \rangle \Leftrightarrow \langle \text{assert} \rangle \\
& \mid \forall \langle \text{var} \rangle .\ \langle \text{assert} \rangle \mid \exists \langle \text{var} \rangle .\ \langle \text{assert} \rangle \\
& \mid \langle \text{exp} \rangle \mapsto \langle \text{exp} \rangle \\
& \mid \mathbf{emptyheap} \\
& \mid \langle \text{assert} \rangle \ * \ \langle \text{assert} \rangle \\
& \mid \langle \text{assert} \rangle \ -\!\!* \ \langle \text{assert} \rangle
\end{aligned}
$$

The Semantics of Assertions

$$[\![-]\!]_{\text{assert}} \in \langle \text{assert} \rangle \to \text{Stores} \to \text{Heaps} \to \mathbf{B}.$$

$$[\![b]\!]_{\text{assert}}\, s\, h \text{ iff } [\![b]\!]_{\text{boolexp}}\, s,$$

$$[\![\neg\, p]\!]_{\text{assert}}\, s\, h \text{ iff } \neg [\![p]\!]_{\text{assert}}\, s\, h,$$

$$[\![p_0 \wedge p_1]\!]_{\text{assert}}\, s\, h \text{ iff } [\![p_0]\!]_{\text{assert}}\, s\, h \wedge [\![p_1]\!]_{\text{assert}}\, s\, h$$

(and similarly for $\vee$, $\Rightarrow$, $\Leftrightarrow$).

$$[\![\forall v.\ p]\!]_{\text{assert}}\, s\, h \text{ iff } \forall x \in \mathbf{Z}.\ [\![p]\!]_{\text{assert}}([\ s \mid v{:}\, x\ ], h),$$

$$[\![\exists v.\ p]\!]_{\text{assert}}\, s\, h \text{ iff } \exists x \in \mathbf{Z}.\ [\![p]\!]_{\text{assert}}([\ s \mid v{:}\, x\ ], h),$$

The Semantics of Assertions (continued)

$$[\![e \mapsto e']\!]_{\text{assert}}\, s\, h \text{ iff}$$
$$\text{dom}\, h = \{[\![e]\!]_{\text{exp}} s\} \text{ and } h([\![e]\!]_{\text{exp}} s) = [\![e']\!]_{\text{exp}} s,$$

$$[\![\textbf{emptyheap}]\!]_{\text{assert}}\, s\, h \text{ iff } \text{dom}\, h = \{\},$$

$$[\![p_0 * p_1]\!]_{\text{assert}}\, s\, h \text{ iff}$$
$$\exists h_0, h_1.\ h_0 \perp h_1 \text{ and } h_0 \cdot h_1 = h \text{ and}$$
$$[\![p_0]\!]_{\text{assert}}\, s\, h_0 \text{ and } [\![p_1]\!]_{\text{assert}}\, s\, h_1,$$

$$[\![p_0 \mathrel{-\!\!*} p_1]\!]_{\text{assert}}\, s\, h \text{ iff}$$
$$\forall h'.\ (h' \perp h \text{ and } [\![p_0]\!]_{\text{assert}}\, s\, h') \text{ implies}$$
$$[\![p_1]\!]_{\text{assert}}\, s\, (h \cdot h'),$$

where

$$h_0 \perp h_1 \text{ iff } \text{dom}\, h_0 \cap \text{dom}\, h_1 = \{\}$$
$$\text{dom}(h_0 \cdot h_1) = \text{dom}\, h_0 \cup \text{dom}\, h_1$$
$$(h_0 \cdot h_1)\ell = \begin{cases} h_0\ell & \text{when } \ell \in \text{dom}\, h_0 \\ h_1\ell & \text{when } \ell \in \text{dom}\, h_1. \end{cases}$$

Examples

$$
\begin{aligned}
[\![\mathsf{x} + \mathsf{y} = \mathsf{x}]\!]_{\text{assert}}\, s\, h \;&\text{ iff }\; [\![\mathsf{x} + \mathsf{y} = \mathsf{x}]\!]_{\text{boolexp}}\, s \\
&\text{iff }\; s\,\mathsf{y} = 0.
\end{aligned}
$$

$$
\begin{aligned}
[\![\forall \mathsf{x}.\ \mathsf{x} + \mathsf{y} = \mathsf{x}]\!]_{\text{assert}}\, s\, h \;&\text{ iff }\; \forall n \in \mathbf{Z}.\ [\![\mathsf{x} + \mathsf{y} = \mathsf{x}]\!]_{\text{assert}}\, [\, s \mid \mathsf{x}\colon n \,]\, h \\
&\text{iff }\; \forall n \in \mathbf{Z}.\ [\, s \mid \mathsf{x}\colon n \,]\, \mathsf{y} = 0 \\
&\text{iff }\; \forall n \in \mathbf{Z}.\ s\,\mathsf{y} = 0 \\
&\text{iff }\; s\,\mathsf{y} = 0.
\end{aligned}
$$

$$
\begin{aligned}
[\![\mathsf{x} \mapsto \mathsf{y}]\!]_{\text{assert}}\, s\, h \;&\text{ iff }\; \operatorname{dom} h = \{s\,\mathsf{x}\} \text{ and } h(s\,\mathsf{x}) = s\,\mathsf{y} \\
&\text{iff }\; h = \{\langle s\,\mathsf{x}, s\,\mathsf{y}\rangle\}.
\end{aligned}
$$

Some Abbreviations

$$e \mapsto - \stackrel{\text{def}}{=} \exists x'.\, e \mapsto x' \quad \text{where } x' \text{ not free in } e$$

$$e \hookrightarrow e' \stackrel{\text{def}}{=} e \mapsto e' \ * \ \mathbf{true}$$

$$e \mapsto e_0, \ldots, e_{n-1} \stackrel{\text{def}}{=} e \mapsto e_0 \ * \ \cdots \ * \ e + n - 1 \mapsto e_{n-1}$$

$$e \hookrightarrow e_0, \ldots, e_{n-1} \stackrel{\text{def}}{=} e \hookrightarrow e_0 \ * \ \cdots \ * \ e + n - 1 \hookrightarrow e_{n-1}$$
$$\text{iff } e \mapsto e_0, \ldots, e_{n-1} \ * \ \mathbf{true}$$

$$e \doteq e' \stackrel{\text{def}}{=} (e = e') \wedge \mathbf{emptyheap}$$

Examples

$$[\![\mathsf{x} \mapsto -]\!]_{\text{assert}} s\, h \ \text{ iff } \ \operatorname{dom} h = \{s\,\mathsf{x}\}$$

$$[\![\mathsf{x} \hookrightarrow \mathsf{y}]\!]_{\text{assert}} s\, h \ \text{ iff } \ s\,\mathsf{x} \in \operatorname{dom} h \text{ and } h(s\,\mathsf{x}) = s\,\mathsf{y}$$

$$[\![\mathsf{x} \hookrightarrow -]\!]_{\text{assert}} s\, h \ \text{ iff } \ s\,\mathsf{x} \in \operatorname{dom} h$$

$$[\![\mathsf{x} \mapsto \mathsf{y}, \mathsf{z}]\!]_{\text{assert}} s\, h \ \text{ iff } \ h = \{\langle s\,\mathsf{x}, s\,\mathsf{y}\rangle, \langle s\,\mathsf{x} + 1, s\,\mathsf{z}\rangle\}$$

$$[\![\mathsf{x} \hookrightarrow \mathsf{y}, \mathsf{z}]\!]_{\text{assert}} s\, h \ \text{ iff } \ h \supseteq \{\langle s\,\mathsf{x}, s\,\mathsf{y}\rangle, \langle s\,\mathsf{x} + 1, s\,\mathsf{z}\rangle\}$$

$$[\![e \doteq e']\!]_{\text{assert}} s\, h \ \text{ iff } \ [\![e]\!]_{\text{exp}} s = [\![e']\!]_{\text{exp}} s \text{ and } \operatorname{dom} h = \{\}$$

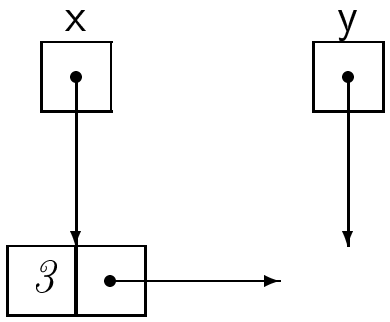
An Example of $*$

$[\![\mathsf{x} \mapsto 3, \mathsf{y} \ * \ \mathsf{y} \mapsto 4, \mathsf{x}]\!]_{\text{assert}}\, s\, h$

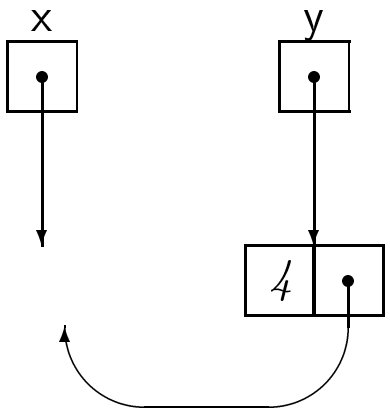
iff $\exists h_0, h_1.\ h_0 \perp h_1$ and $h_0 \cdot h_1 = h$

and $[\![\mathsf{x} \mapsto 3, \mathsf{y}]\!]_{\text{assert}}\, s\, h_0$ and $[\![\mathsf{y} \mapsto 4, \mathsf{x}]\!]_{\text{assert}}\, s\, h_1$

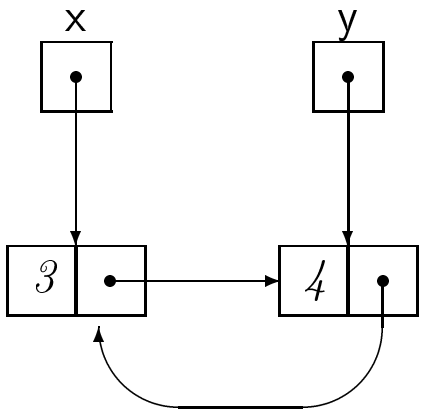
$\mathsf{x} \mapsto 3, \mathsf{y}$



$\mathsf{y} \mapsto 4, \mathsf{x}$



$\mathsf{x} \mapsto 3, \mathsf{y} \ * \ \mathsf{y} \mapsto 4, \mathsf{x}$



More Examples of $*$

Suppose $s\,\mathsf{x}$ and $s\,\mathsf{y}$ are distinct locations, so that

$$h_1 = \{\langle s\,\mathsf{x}, 1\rangle\} \qquad \text{and} \qquad h_2 = \{\langle s\,\mathsf{y}, 2\rangle\}.$$

are heaps with disjoint domains. Then

If P is:	then $[\![P]\!]s\,h$ is:
$\mathsf{x} \mapsto 1$	$h = h_1$
$\mathsf{y} \mapsto 2$	$h = h_2$
$\mathsf{x} \mapsto 1 \ * \ \mathsf{y} \mapsto 2$	$h = h_1 \cdot h_2$
$\mathsf{x} \mapsto 1 \ * \ \mathsf{x} \mapsto 1$	**false**
$\mathsf{x} \mapsto 1 \vee \mathsf{y} \mapsto 2$	$h = h_1$ or $h = h_2$
$\mathsf{x} \mapsto 1 \ * \ (\mathsf{x} \mapsto 1 \vee \mathsf{y} \mapsto 2)$	$h = h_1 \cdot h_2$
$(\mathsf{x} \mapsto 1 \vee \mathsf{y} \mapsto 2) \ * \ (\mathsf{x} \mapsto 1 \vee \mathsf{y} \mapsto 2)$	$h = h_1 \cdot h_2$
$\mathsf{x} \mapsto 1 \ * \ \mathsf{y} \mapsto 2 \ * \ (\mathsf{x} \mapsto 1 \vee \mathsf{y} \mapsto 2)$	**false**
$\mathsf{x} \mapsto 1 \ * \ \mathbf{true}$	$h_1 \subseteq h$
$\mathsf{x} \mapsto 1 \ * \ \neg\, \mathsf{x} \mapsto 1$	$h_1 \subseteq h$

An Example of $-\!\!*$

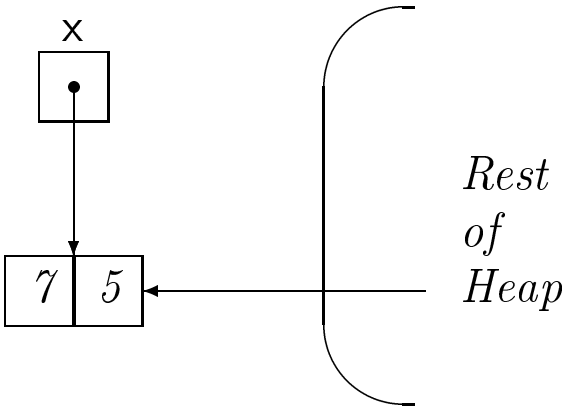
$$[\![\mathsf{x} \mapsto 7,5 \mathrel{-\!\!*} P]\!]_{\text{assert}}\, s\, h$$

iff $\forall h'.\ (h' \perp h$ and $[\![\mathsf{x} \mapsto 7,5]\!]_{\text{assert}}\, s\, h')$

implies $[\![P]\!]_{\text{assert}}\, s\, (h \cdot h')$

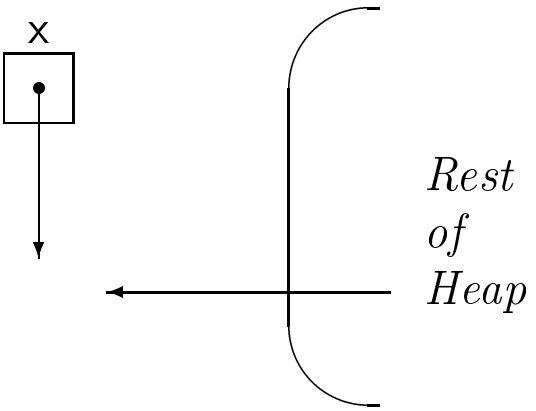
If:

P



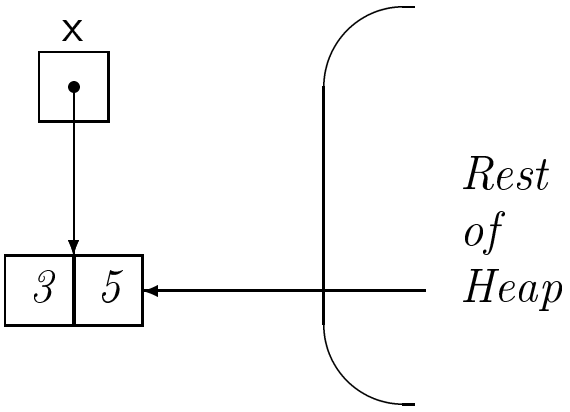
then:

$\mathsf{x} \mapsto 7,5 \mathrel{-\!\!*} P$



and:

$\mathsf{x} \mapsto 3,5 \ * \ (\mathsf{x} \mapsto 7,5 \mathrel{-\!\!*} P)$



Some Terminology

- When $[\![p]\!]_{\text{assert}}\, s\, h = \mathbf{true}$ we say
  - p is *true* in (s, h),
  - p *holds* for (s, h),
  - p *describes* (s, h),
  - (s, h) *satisfies* p.
- When $[\![p]\!]_{\text{assert}}\, s\, h = \mathbf{true}$ for all states $(s, h) \in$ States, we say
  - p is *valid*.
- When $[\![p]\!]_{\text{assert}}\, s\, h = \mathbf{false}$ for all states $(s, h) \in$ States, we say
  - p is *unsatisfiable*.
- When $[\![q]\!]_{\text{assert}}\, s\, h = \mathbf{true}$ holds for every state $(s, h) \in$ States such that $[\![p]\!]_{\text{assert}}\, s\, h = \mathbf{true}$, we say
  - p is *stronger* than q,
  - q is *weaker* than p.

  (Notice that **true** is weaker than any assertion and **false** is stronger than any assertion.)
- When $[\![q]\!]_{\text{assert}}\, s\, h = \mathbf{true}$ holds for exactly those states $(s, h) \in$ States such that $[\![p]\!]_{\text{assert}}\, s\, h = \mathbf{true}$, we say
  - p is *equivalent* to q.

Substitution

Proposition 6 *Let δ abbreviate the substitution*

$$v_1 \to e_1, \ldots, v_n \to e_n,$$

and for any store s, let

$$\widehat{s} = [\, v_1 \colon [\![e_1]\!]_{\mathrm{exp}} s \mid \ldots \mid v_n \colon [\![e_n]\!]_{\mathrm{exp}} s \,].$$

If p is an assertion whose free variables belong to $\{v_1, \ldots, v_n\}$, then

$$[\![p/\delta]\!]_{\mathrm{assert}} s\, h = [\![p]\!]_{\mathrm{assert}} \widehat{s}\, h.$$

Eventually, δ will map

ordinary variables	into	expressions
sequence variables	into	sequence expressions
tree variables	into	tree expressions
assertion variables	into	assertions

$$\vdots$$

Inference Rules

- An *inference rule* consists of zero or more *premisses* and a single *conclusion*, separated by a horizontal line:

$$\frac{\mathcal{P}_1 \qquad \cdots \qquad \mathcal{P}_n}{\mathcal{C}}$$

- The premisses and conclusion are schemata, i.e., they may contain *metavariables*, each of which ranges over some set of phrases.

- An instance of an inference rule is obtained by replacing each metavariable by a phrase in its range (satisfying any *side-conditions*). In an instance, the premisses and conclusion will be assertions (or transitions or specifications).

- A *formal proof* is a sequence of assertions, each of which is the conclusion of some instance of an inference rule whose premisses occur earlier in the sequence.

- An inference rule with zero premisses is called an *axiom schema*. The overbar is sometimes omitted.

- An axiom schema containing no metavariables is called an *axiom*. The overbar is usually omitted.

Some Inference Rules

$$\mathsf{x} + 0 = \mathsf{x} \qquad \text{(an axiom)}$$

$$\frac{}{e_1 = e_0 \Rightarrow e_0 = e_1} \qquad \text{(an axiom schema)}$$

$$\frac{p \qquad p \Rightarrow q}{q}$$

$$\frac{p}{\forall v.\ p}$$

Some Instances

$$\mathsf{x} + 0 = \mathsf{x}$$

$$\frac{}{\mathsf{x} + 0 = \mathsf{x} \Rightarrow \mathsf{x} = \mathsf{x} + 0}$$

$$\frac{\mathsf{x} + 0 = \mathsf{x} \qquad \mathsf{x} + 0 = \mathsf{x} \Rightarrow \mathsf{x} = \mathsf{x} + 0}{\mathsf{x} = \mathsf{x} + 0}$$

$$\frac{\mathsf{x} = \mathsf{x} + 0}{\forall \mathsf{x}.\ \mathsf{x} = \mathsf{x} + 0}$$

A Formal Proof

1.	$\mathsf{x} + 0 = \mathsf{x}$	(First rule)
2.	$\mathsf{x} + 0 = \mathsf{x} \Rightarrow \mathsf{x} = \mathsf{x} + 0$	(Second rule)
3.	$\mathsf{x} = \mathsf{x} + 0$	(Third rule, 1, 2)
4.	$\forall \mathsf{x}.\ \mathsf{x} = \mathsf{x} + 0$	(Fourth rule, 3)

$$\dfrac{\dfrac{}{\mathsf{x} + 0 = \mathsf{x}} \qquad \dfrac{}{\mathsf{x} + 0 = \mathsf{x} \Rightarrow \mathsf{x} = \mathsf{x} + 0}}{\dfrac{\mathsf{x} = \mathsf{x} + 0}{\forall \mathsf{x}.\ \mathsf{x} = \mathsf{x} + 0}}$$

Soundness

An inference rule is *sound* iff, for all instances,

- if the premisses of the instance are all valid (i.e. true in all states),
- then the conclusion is valid (i.e. true in all states).

(It follows that the assertions in a formal proof must all be valid.)

For example,

$$\frac{p}{\forall v.\ p}$$

is sound, since for any assertion p and variable v, if p is true in all states, then $\forall v.\ p$ is true in all states.

However,

$$\frac{}{p \Rightarrow \forall v.\ p}$$

is not sound, since its instance

$$\frac{}{\mathsf{x} > 0 \Rightarrow \forall \mathsf{x}.\ \mathsf{x} > 0}$$

has a conclusion that does not hold in a state that maps x into 3.

Some Inference Rules for $\Rightarrow$ and $\neg$

$$\frac{p \qquad p \Rightarrow q}{q} \qquad \text{(Modus Ponens)}$$

$$p \Rightarrow (q \Rightarrow p)$$

$$(p \Rightarrow (q \Rightarrow r)) \Rightarrow ((p \Rightarrow q) \Rightarrow (p \Rightarrow r))$$

$$(\neg q \Rightarrow \neg p) \Rightarrow ((\neg q \Rightarrow p) \Rightarrow q)$$

Other Connectives

$$(p \Leftrightarrow q) \Rightarrow (p \Rightarrow q)$$

$$(p \Leftrightarrow q) \Rightarrow (q \Rightarrow p)$$

$$(p \wedge q) \Leftrightarrow \neg(p \Rightarrow \neg q)$$

$$(p \vee q) \Leftrightarrow (\neg p) \Rightarrow q$$

$$(p \Leftrightarrow q) \Leftrightarrow (p \Rightarrow q) \wedge (q \Rightarrow p)$$

Quantifiers

$$\frac{p}{\forall v.\ p} \qquad \text{(Generalization)}$$

$$(\forall v.\ p) \Rightarrow (p/v \rightarrow e)$$

$$(\forall v.\ p \Rightarrow q) \Leftrightarrow (p \Rightarrow \forall v.\ q) \qquad \text{when } v \text{ is not free in } p$$

$$(\exists v.\ p) \Leftrightarrow \neg\,(\forall v.\ \neg p)$$

$$(p/v \rightarrow e) \Rightarrow (\exists v.\ p)$$

$$(\exists v.\ p \Rightarrow q) \Leftrightarrow (p \Rightarrow \exists v.\ q) \qquad \text{when } v \text{ is not free in } p$$

Equality

$$e = e$$

$$e = e' \Rightarrow e' = e$$

$$e = e' \wedge e' = e'' \Rightarrow e = e''$$

$$e_1 = e_2 \Rightarrow (e/v \rightarrow e_1) = (e/v \rightarrow e_2)$$

Rules and Axiom Schemata for $*$ and $-\!\!*$

$$p_1 * p_2 \Leftrightarrow p_2 * p_1$$

$$(p_1 * p_2) * p_3 \Leftrightarrow p_1 * (p_2 * p_3)$$

$$p * \mathbf{emptyheap} \Leftrightarrow p$$

$$(p_1 \vee p_2) * q \Leftrightarrow (p_1 * q) \vee (p_2 * q)$$

$$(p_1 \wedge p_2) * q \Rightarrow (p_1 * q) \wedge (p_2 * q)$$

$$(\exists x.\, p_1) * p_2 \Leftrightarrow \exists x.\, (p_1 * p_2) \quad \text{when } x \text{ not free in } p_2$$

$$(\forall x.\, p_1) * p_2 \Rightarrow \forall x.\, (p_1 * p_2) \quad \text{when } x \text{ not free in } p_2$$

$$\frac{p_1 \Rightarrow p_2 \qquad q_1 \Rightarrow q_2}{p_1 * q_1 \Rightarrow p_2 * q_2}$$

$$\frac{p_1 * p_2 \Rightarrow p_3}{p_1 \Rightarrow (p_2 -\!\!* p_3)} \qquad \frac{p_1 \Rightarrow (p_2 -\!\!* p_3)}{p_1 * p_2 \Rightarrow p_3.}$$

Axiom Schemata for $\mapsto$

$$x \mapsto e \wedge x' \mapsto e' \Leftrightarrow x \mapsto e \wedge x = x' \wedge e = e'$$

$$x \mapsto e * x' \mapsto e' * \mathbf{true} \Rightarrow x \neq x'$$

$$\mathbf{emptyheap} \Leftrightarrow \forall x.\ \neg(x \mapsto - * \mathbf{true})$$

$$(e \hookrightarrow x) \wedge p \Rightarrow (e \mapsto x) * ((e \mapsto x) -\!\!* p).$$

Pure Assertions

An assertion is *pure* if it does not contain $\mapsto$ or **emptyheap**.

A pure assertion does not depend upon the heap.

Axiom Schemata for Purity

$$p_1 \wedge p_2 \Rightarrow p_1 \ * \ p_2 \quad \text{when } p_1 \text{ or } p_2 \text{ is pure}$$

$$p_1 \ * \ p_2 \Rightarrow p_1 \wedge p_2 \quad \text{when } p_1 \text{ and } p_2 \text{ are pure}$$

$$(p \wedge q) \ * \ r \Leftrightarrow (p \ * \ r) \wedge q \qquad \text{when } q \text{ is pure}$$

$$(p_1 \mathrel{-\!\!*} p_2) \Rightarrow (p_1 \Rightarrow p_2) \quad \text{when } p_1 \text{ is pure}$$

$$(p_1 \Rightarrow p_2) \Rightarrow (p_1 \mathrel{-\!\!*} p_2) \quad \text{when } p_1 \text{ and } p_2 \text{ are pure}$$

In addition, the model validates all of the laws of classical logic for $\neg$, $\Rightarrow$, $\wedge$ and $\vee$. It uses the boolean algebra structure of the powerset of Heaps, which amounts to pointwise interpretations for each classical connective.

Two Unsound Axiom Schemata

$$p \not\Rrightarrow p * p \qquad \text{(Contraction)}$$

e.g. $p : \mathsf{x} \mapsto 1$

$$p * q \not\Rrightarrow p \qquad \text{(Weakening)}$$

e.g. $p : \mathsf{x} \mapsto 1$

$q : \mathsf{y} \mapsto 2$

Some Derived Inference Rules

$$\frac{}{q \ * \ (q \multimap p) \Rightarrow p}$$

Derivation:

$$\frac{q \ * \ (q \multimap p) \Rightarrow (q \multimap p) \ * \ q \qquad \dfrac{(q \multimap p) \Rightarrow (q \multimap p)}{(q \multimap p) \ * \ q \Rightarrow p}}{q \ * \ (q \multimap p) \Rightarrow p}$$

$$\frac{}{r \Rightarrow (q \multimap (q \ * \ r))}$$

Deriviation:

$$\frac{(r \ * \ q) \Rightarrow (q \ * \ r)}{r \Rightarrow (q \multimap (q \ * \ r))}$$

$$\frac{}{(p \ * \ r) \Rightarrow (p \ * \ (q \multimap (q \ * \ r)))}$$

Derivation:

$$\frac{p \Rightarrow p \qquad r \Rightarrow (q \multimap (q \ * \ r))}{(p \ * \ r) \Rightarrow (p \ * \ (q \multimap (q \ * \ r)))}$$

More Derived Inference Rules

$$\frac{p_1 \Rightarrow (q \mathrel{-\!\!*} r) \qquad p_2 \Rightarrow (r \mathrel{-\!\!*} s)}{p_2 \ * \ p_1 \Rightarrow (q \mathrel{-\!\!*} s)}$$

Derivation:

$$\dfrac{\dfrac{\dfrac{}{p_2 \Rightarrow p_2} \qquad \dfrac{p_1 \Rightarrow (q \mathrel{-\!\!*} r)}{p_1 \ * \ q \Rightarrow r}}{p_2 \ * \ p_1 \ * \ q \Rightarrow p_2 \ * \ r} \qquad \dfrac{p_2 \Rightarrow (r \mathrel{-\!\!*} s)}{p_2 \ * \ r \Rightarrow s}}{\dfrac{p_2 \ * \ p_1 \ * \ q \Rightarrow s}{p_2 \ * \ p_1 \Rightarrow (q \mathrel{-\!\!*} s)}}$$

$$\frac{p' \Rightarrow p \qquad q \Rightarrow q'}{(p \mathrel{-\!\!*} q) \Rightarrow (p' \mathrel{-\!\!*} q').}$$

Derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{(p \mathrel{-\!\!*} q) \Rightarrow (p \mathrel{-\!\!*} q)} \qquad p' \Rightarrow p}{(p \mathrel{-\!\!*} q) \ * \ p' \Rightarrow (p \mathrel{-\!\!*} q) \ * \ p} \qquad \dfrac{\dfrac{}{(p \mathrel{-\!\!*} q) \Rightarrow (p \mathrel{-\!\!*} q)}}{(p \mathrel{-\!\!*} q) \ * \ p \Rightarrow q}}{(p \mathrel{-\!\!*} q) \ * \ p' \Rightarrow q} \qquad q \Rightarrow q'}{\dfrac{(p \mathrel{-\!\!*} q) \ * \ p' \Rightarrow q'}{(p \mathrel{-\!\!*} q) \Rightarrow (p' \mathrel{-\!\!*} q')}}$$

Specifications

$$\langle \text{spec} \rangle ::= \{\langle \text{assert} \rangle\}\ \langle \text{comm} \rangle\ \{\langle \text{assert} \rangle\} \quad \text{(partial correctness)}$$
$$\mid [\ \langle \text{assert} \rangle\ ]\ \langle \text{comm} \rangle\ [\ \langle \text{assert} \rangle\ ] \quad \text{(total correctness)}$$

$$[\![-]\!]_{\text{spec}} \in \langle \text{spec} \rangle \to \mathbf{B}.$$

Partial correctness:

$[\![\{p\}\ c\ \{q\}]\!]_{\text{spec}}$ iff $\forall (s,h) \in \text{States}.\ [\![p]\!]_{\text{assert}}\, s\, h$ implies

$\neg\,(c, (s,h) \rightsquigarrow^* \textbf{abort})$

and $(\forall (s',h') \in \text{States}.\ c, (s,h) \rightsquigarrow^* (s',h')$ implies $[\![q]\!]_{\text{assert}}\, s'\, h')$

Total correctness:

$[\![[\ p\ ]\ c\ [\ q\ ]]\!]_{\text{spec}}$ iff $\forall (s,h) \in \text{States}.\ [\![p]\!]_{\text{assert}}\, s\, h$ implies

$\neg\,(c, (s,h) \rightsquigarrow^* \textbf{abort})$

and $\neg\,(c, (s,h)\uparrow)$

and $(\forall (s',h') \in \text{States}.\ c, (s,h) \rightsquigarrow^* (s',h')$ implies $[\![q]\!]_{\text{assert}}\, s'\, h')$.

- Specifications are implicitly quantified over both stores and heaps — and over all executions.
- An execution giving a memory fault falsifies a specification.

Examples of Partial Correctness Specifications

$$\{\mathsf{x} - \mathsf{y} > 3\}\ \mathsf{x} := \mathsf{x} - \mathsf{y}\ \{\mathsf{x} > 3\}$$

$$\{\mathsf{x} + \mathsf{y} \geq 17\}\ \mathsf{x} := \mathsf{x} + 10\ \{\mathsf{x} + \mathsf{y} \geq 27\}$$

$$\{\mathsf{x} \leq 10\}\ \textbf{while}\ \mathsf{x} \neq 10\ \textbf{do}\ \mathsf{x} := \mathsf{x} + 1\ \{\mathsf{x} = 10\}$$

$$\{\textbf{true}\}\ \textbf{while}\ \mathsf{x} \neq 10\ \textbf{do}\ \mathsf{x} := \mathsf{x} + 1\ \{\mathsf{x} = 10\}$$

$$\{\mathsf{x} > 10\}\ \textbf{while}\ \mathsf{x} \neq 10\ \textbf{do}\ \mathsf{x} := \mathsf{x} + 1\ \{\textbf{false}\}$$

$$\{\textbf{emptyheap}\}\ \mathsf{x} := \textbf{cons}(1, 2)\ \{\mathsf{x} \mapsto 1, 2\}$$

$$\{\mathsf{x} \mapsto 1, 2\}\ \mathsf{y} := [\mathsf{x}]\ ;\ [\mathsf{x} + 1] := 3\ \{\mathsf{x} \mapsto 1, 3 \wedge \mathsf{y} = 1\}$$

Examples of Total Correctness Specifications

$$[\ \mathsf{x} - \mathsf{y} > 3\ ]\ \mathsf{x} := \mathsf{x} - \mathsf{y}\ [\ \mathsf{x} > 3\ ]$$

$$[\ \mathsf{x} + \mathsf{y} \geq 17\ ]\ \mathsf{x} := \mathsf{x} + 10\ [\ \mathsf{x} + \mathsf{y} \geq 27\ ]$$

$$[\ \mathsf{x} \leq 10\ ]\ \textbf{while}\ \mathsf{x} \neq 10\ \textbf{do}\ \mathsf{x} := \mathsf{x} + 1\ [\ \mathsf{x} = 10\ ]$$

$$[\ \textbf{emptyheap}\ ]\ \mathsf{x} := \textbf{cons}(1, 2)\ [\ \mathsf{x} \mapsto 1, 2\ ]$$

$$[\ \mathsf{x} \mapsto 1, 2\ ]\ \mathsf{y} := [\mathsf{x}]\ ;\ [\mathsf{x} + 1] := 3\ [\ \mathsf{x} \mapsto 1, 3 \wedge \mathsf{y} = 1\ ]$$

Violating Record Boundaries

Suppose P holds for a state (s, h), and C' does not assign to x. Then

$$[\, P \,]\ C\ ;\ \mathsf{x} := \mathbf{cons}(1, 2)\ ;\ C'\ ;\ [\mathsf{x} + 2] := 3\ ;\ C''\ [\, Q \,]$$

cannot hold. For suppose the contrary. Then

$$\langle C, (s, h)\rangle \rightsquigarrow^* (s', h').$$

Let ℓ be such that $\ell, \ell + 1, \ell + 2 \notin \operatorname{dom} h'$. Then

$$\langle \mathsf{x} := \mathbf{cons}(1, 2), (s', h')\rangle \rightsquigarrow ([\, s' \mid \mathsf{x}\colon \ell \,], [\, h' \mid \ell\colon 1 \mid \ell + 1\colon 2 \,]),$$

and

$$\langle C', ([\, s' \mid \mathsf{x}\colon \ell \,], [\, h' \mid \ell\colon 1 \mid \ell + 1\colon 2 \,])\rangle \rightsquigarrow^* (s'', h''),$$

where $s''\mathsf{x} = \ell$ and $\ell + 2 \notin \operatorname{dom} h''$. Then

$$\langle [\mathsf{x} + 2] := 3, (s'', h'')\rangle \rightsquigarrow \mathbf{abort},$$

and

$$\langle [\mathsf{x} + 2] := 3\ ;\ C'', (s'', h'')\rangle \rightsquigarrow \mathbf{abort},$$

contradicting the assumption.

Inference Rules for Specifications:

Assignment (AS)

$$\frac{}{\{p/v \to e\}\ v := e\ \{p\}} \qquad \frac{}{[\,p/v \to e\,]\ v := e\ [\,p\,]}$$

Instances

$$\frac{}{\{\mathsf{x} - \mathsf{y} > 3\}\ \mathsf{x} := \mathsf{x} - \mathsf{y}\ \{\mathsf{x} > 3\}}$$

$$\frac{}{\{(2 \times \mathsf{y}) - \mathsf{y} > 3\}\ \mathsf{x} := 2 \times \mathsf{y}\ \{\mathsf{x} - \mathsf{y} > 3\}}$$

Inference Rules for Specifications:

Sequential Composition (SQ)

$$\frac{\{p\}\ c_1\ \{q\} \qquad \{q\}\ c_2\ \{r\}}{\{p\}\ c_1\ ;\ c_2\ \{r\}} \qquad \frac{[\,p\,]\ c_1\ [\,q\,] \qquad [\,q\,]\ c_2\ [\,r\,]}{[\,p\,]\ c_1\ ;\ c_2\ [\,r\,]}$$

An Instance

$$\frac{\{(2 \times \mathsf{y}) - \mathsf{y} > 3\}\ \mathsf{x} := 2 \times \mathsf{y}\ \{\mathsf{x} - \mathsf{y} > 3\} \qquad \{\mathsf{x} - \mathsf{y} > 3\}\ \mathsf{x} := \mathsf{x} - \mathsf{y}\ \{\mathsf{x} > 3\}}{\{(2 \times \mathsf{y}) - \mathsf{y} > 3\}\ \mathsf{x} := 2 \times \mathsf{y}\ ;\ \mathsf{x} := \mathsf{x} - \mathsf{y}\ \{\mathsf{x} > 3\}}$$

Note

Except for **while** commands and recursive procedures, reasoning for partial and total correctness will be the same, and we will henceforth only give rules for partial correctness.

Inference Rules for Specifications:

Strengthening Precedent (SP)

$$\frac{p \Rightarrow q \qquad \{q\}\ c\ \{r\}}{\{p\}\ c\ \{r\}}$$

An Instance

$$\frac{\mathsf{y} > 3 \Rightarrow (2 \times \mathsf{y}) - \mathsf{y} > 3 \qquad \{(2 \times \mathsf{y}) - \mathsf{y} > 3\}\ \mathsf{x} := 2 \times \mathsf{y}\ ;\ \mathsf{x} := \mathsf{x} - \mathsf{y}\ \{\mathsf{x} > 3\}}{\{\mathsf{y} > 3\}\ \mathsf{x} := 2 \times \mathsf{y}\ ;\ \mathsf{x} := \mathsf{x} - \mathsf{y}\ \{\mathsf{x} > 3\}}$$

Weakening Consequent (WC)

$$\frac{\{p\}\ c\ \{q\} \qquad q \Rightarrow r}{\{p\}\ c\ \{r\}}$$

An Instance

$$\frac{\{\mathsf{y} > 3\}\ \mathsf{x} := 2 \times \mathsf{y}\ ;\ \mathsf{x} := \mathsf{x} - \mathsf{y}\ \{\mathsf{x} > 3\} \qquad \mathsf{x} > 3 \Rightarrow \mathsf{x} \geq 4}{\{\mathsf{y} > 3\}\ \mathsf{x} := 2 \times \mathsf{y}\ ;\ \mathsf{x} := \mathsf{x} - \mathsf{y}\ \{\mathsf{x} \geq 4\}}$$

A Simple Proof

1. $\{(2 \times \mathsf{y}) - \mathsf{y} > 3\}\ \mathsf{x} := 2 \times \mathsf{y}\ \{\mathsf{x} - \mathsf{y} > 3\}$ (AS)
2. $\{\mathsf{x} - \mathsf{y} > 3\}\ \mathsf{x} := \mathsf{x} - \mathsf{y}\ \{\mathsf{x} > 3\}$ (AS)
3. $\{(2 \times \mathsf{y}) - \mathsf{y} > 3\}\ \mathsf{x} := 2 \times \mathsf{y}\ ;\ \mathsf{x} := \mathsf{x} - \mathsf{y}\ \{\mathsf{x} > 3\}$ (SQ,1,2)
4. $\mathsf{y} > 3 \Rightarrow (2 \times \mathsf{y}) - \mathsf{y} > 3$ (VC)
5. $\{\mathsf{y} > 3\}\ \mathsf{x} := 2 \times \mathsf{y}\ ;\ \mathsf{x} := \mathsf{x} - \mathsf{y}\ \{\mathsf{x} > 3\}$ (SP,4,3)
6. $\mathsf{x} > 3 \Rightarrow \mathsf{x} \geq 4$ (VC)
7. $\{\mathsf{y} > 3\}\ \mathsf{x} := 2 \times \mathsf{y}\ ;\ \mathsf{x} := \mathsf{x} - \mathsf{y}\ \{\mathsf{x} \geq 4\}$ (WC,5,6)

A More Realistic Proof

1. $\{\mathsf{f} = \mathrm{fib}(\mathsf{k}+1) \wedge \mathsf{g} = \mathrm{fib}((\mathsf{k}+1)-1) \wedge \mathsf{k}+1 \leq \mathsf{n}\}$
   $\mathsf{k} := \mathsf{k}+1$
   $\{\mathsf{f} = \mathrm{fib}(\mathsf{k}) \wedge \mathsf{g} = \mathrm{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n}\}$ (AS)
2. $\{\mathsf{f}+\mathsf{t} = \mathrm{fib}(\mathsf{k}+1) \wedge \mathsf{g} = \mathrm{fib}((\mathsf{k}+1)-1) \wedge \mathsf{k}+1 \leq \mathsf{n}\}$
   $\mathsf{f} := \mathsf{f}+\mathsf{t}$
   $\{\mathsf{f} = \mathrm{fib}(\mathsf{k}+1) \wedge \mathsf{g} = \mathrm{fib}((\mathsf{k}+1)-1) \wedge \mathsf{k}+1 \leq \mathsf{n}\}$ (AS)
3. $\{\mathsf{f}+\mathsf{t} = \mathrm{fib}(\mathsf{k}+1) \wedge \mathsf{g} = \mathrm{fib}((\mathsf{k}+1)-1) \wedge \mathsf{k}+1 \leq \mathsf{n}\}$
   $\mathsf{f} := \mathsf{f}+\mathsf{t}\ ;\ \mathsf{k} := \mathsf{k}+1$
   $\{\mathsf{f} = \mathrm{fib}(\mathsf{k}) \wedge \mathsf{g} = \mathrm{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n}\}$ (SQ,1,2)
4. $\{\mathsf{f}+\mathsf{t} = \mathrm{fib}(\mathsf{k}+1) \wedge \mathsf{f} = \mathrm{fib}((\mathsf{k}+1)-1) \wedge \mathsf{k}+1 \leq \mathsf{n}\}$
   $\mathsf{g} := \mathsf{f}$
   $\{\mathsf{f}+\mathsf{t} = \mathrm{fib}(\mathsf{k}+1) \wedge \mathsf{g} = \mathrm{fib}((\mathsf{k}+1)-1) \wedge \mathsf{k}+1 \leq \mathsf{n}\}$ (AS)
5. $\{\mathsf{f}+\mathsf{t} = \mathrm{fib}(\mathsf{k}+1) \wedge \mathsf{f} = \mathrm{fib}((\mathsf{k}+1)-1) \wedge \mathsf{k}+1 \leq \mathsf{n}\}$
   $\mathsf{g} := \mathsf{f}\ ;\ \mathsf{f} := \mathsf{f}+\mathsf{t}\ ;\ \mathsf{k} := \mathsf{k}+1$
   $\{\mathsf{f} = \mathrm{fib}(\mathsf{k}) \wedge \mathsf{g} = \mathrm{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n}\}$ (SQ,3,4)

A More Realistic Proof (continued)

6. $\{\mathsf{f} + \mathsf{g} = \text{fib}(\mathsf{k}+1) \wedge \mathsf{f} = \text{fib}((\mathsf{k}+1)-1) \wedge \mathsf{k}+1 \leq \mathsf{n}\}$
   $\mathsf{t} := \mathsf{g}$
   $\{\mathsf{f} + \mathsf{t} = \text{fib}(\mathsf{k}+1) \wedge \mathsf{f} = \text{fib}((\mathsf{k}+1)-1) \wedge \mathsf{k}+1 \leq \mathsf{n}\}$ (AS)

7. $\{\mathsf{f} + \mathsf{g} = \text{fib}(\mathsf{k}+1) \wedge \mathsf{f} = \text{fib}((\mathsf{k}+1)-1) \wedge \mathsf{k}+1 \leq \mathsf{n}\}$
   $\mathsf{t} := \mathsf{g}\ ;\ \mathsf{g} := \mathsf{f}\ ;\ \mathsf{f} := \mathsf{f} + \mathsf{t}\ ;\ \mathsf{k} := \mathsf{k}+1$
   $\{\mathsf{f} = \text{fib}(\mathsf{k}) \wedge \mathsf{g} = \text{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n}\}$ (SQ,5,6)

8. $(\mathsf{f} = \text{fib}(\mathsf{k}) \wedge \mathsf{g} = \text{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n} \wedge \mathsf{k} \neq \mathsf{n}) \Rightarrow$
   $(\mathsf{f} + \mathsf{g} = \text{fib}(\mathsf{k}+1) \wedge \mathsf{f} = \text{fib}((\mathsf{k}+1)-1) \wedge \mathsf{k}+1 \leq \mathsf{n})$
   (VC*)

9. $\{\mathsf{f} = \text{fib}(\mathsf{k}) \wedge \mathsf{g} = \text{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n} \wedge \mathsf{k} \neq \mathsf{n}\}$
   $\mathsf{t} := \mathsf{g}\ ;\ \mathsf{g} := \mathsf{f}\ ;\ \mathsf{f} := \mathsf{f} + \mathsf{t}\ ;\ \mathsf{k} := \mathsf{k}+1$
   $\{\mathsf{f} = \text{fib}(\mathsf{k}) \wedge \mathsf{g} = \text{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n}\}$ (SP,7,8)

*since

$$\text{fib}(0) = 0 \qquad \text{fib}(1) = 1 \qquad \text{fib}(i) = \text{fib}(i-1) + \text{fib}(i-2).$$

Inference Rules for Specifications:

Partial Correctness of **while** (WHP)

$$\frac{\{i \wedge b\}\ c\ \{i\}}{\{i\}\ \mathbf{while}\ b\ \mathbf{do}\ c\ \{i \wedge \neg\, b\}}$$

Here i is the *invariant*.

An Instance

$$\frac{\begin{array}{l}\{\mathsf{f} = \mathrm{fib}(\mathsf{k}) \wedge \mathsf{g} = \mathrm{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n} \wedge \mathsf{k} \neq \mathsf{n}\} \\ \mathsf{t} := \mathsf{g}\ ;\ \mathsf{g} := \mathsf{f}\ ;\ \mathsf{f} := \mathsf{f} + \mathsf{t}\ ;\ \mathsf{k} := \mathsf{k} + 1 \\ \{\mathsf{f} = \mathrm{fib}(\mathsf{k}) \wedge \mathsf{g} = \mathrm{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n}\}\end{array}}{\begin{array}{l}\{\mathsf{f} = \mathrm{fib}(\mathsf{k}) \wedge \mathsf{g} = \mathrm{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n}\} \\ \mathbf{while}\ \mathsf{k} \neq \mathsf{n}\ \mathbf{do}\ (\mathsf{t} := \mathsf{g}\ ;\ \mathsf{g} := \mathsf{f}\ ;\ \mathsf{f} := \mathsf{f} + \mathsf{t}\ ;\ \mathsf{k} := \mathsf{k} + 1) \\ \{\mathsf{f} = \mathrm{fib}(\mathsf{k}) \wedge \mathsf{g} = \mathrm{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n} \wedge \neg\, \mathsf{k} \neq \mathsf{n}\}\end{array}}$$

Inference Rules for Specifications:

Total Correctness of **while** (WHT)

$$\frac{[\, i \wedge b \wedge e = v_0 \,]\ c\ [\, i \wedge e < v_0 \,] \qquad (i \wedge b) \Rightarrow e \geq 0}{[\, i \,]\ \mathbf{while}\ b\ \mathbf{do}\ c\ [\, i \wedge \neg\ b \,]}$$

when v_0 does not occur free in i, b, c, or e.
Here i is the *invariant*, e is the *variant*, and v_0 is a *ghost variable*.

An Instance

$$\frac{\begin{array}{l}[\mathsf{f} = \mathrm{fib}(\mathsf{k}) \wedge \mathsf{g} = \mathrm{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n} \wedge \mathsf{k} \neq \mathsf{n} \wedge \mathsf{n} - \mathsf{k} = \mathsf{v}_0] \\ \mathsf{t} := \mathsf{g}\ ;\ \mathsf{g} := \mathsf{f}\ ;\ \mathsf{f} := \mathsf{f} + \mathsf{t}\ ;\ \mathsf{k} := \mathsf{k} + 1 \\ [\mathsf{f} = \mathrm{fib}(\mathsf{k}) \wedge \mathsf{g} = \mathrm{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n} \wedge \mathsf{n} - \mathsf{k} < \mathsf{v}_0]\end{array} \qquad (\mathsf{f} = \mathrm{fib}(\mathsf{k}) \wedge \mathsf{g} = \mathrm{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n} \wedge \mathsf{k} \neq \mathsf{n}) \Rightarrow \mathsf{n} - \mathsf{k} \geq 0}{\begin{array}{l}[\mathsf{f} = \mathrm{fib}(\mathsf{k}) \wedge \mathsf{g} = \mathrm{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n}] \\ \mathbf{while}\ \mathsf{k} \neq \mathsf{n}\ \mathbf{do}\ (\mathsf{t} := \mathsf{g}\ ;\ \mathsf{g} := \mathsf{f}\ ;\ \mathsf{f} := \mathsf{f} + \mathsf{t}\ ;\ \mathsf{k} := \mathsf{k} + 1) \\ [\mathsf{f} = \mathrm{fib}(\mathsf{k}) \wedge \mathsf{g} = \mathrm{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n} \wedge \neg\, \mathsf{k} \neq \mathsf{n}]\end{array}}$$

Inference Rules for Specifications:

skip (SK)

$$\frac{}{\{p\}\ \mathbf{skip}\ \{p\}}$$

Conditional (CD)

$$\frac{\{p \wedge b\}\ c_0\ \{q\} \qquad \{p \wedge \neg\ b\}\ c_1\ \{q\}}{\{p\}\ \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1\ \{q\}}$$

Inference Rules for Specifications:

Variable Declaration (DC)

$$\frac{\{p\}\ c\ \{q\}}{\{p\}\ \textbf{newvar}\ v\ \textbf{in}\ c\ \{q\}}$$

when v does not occur free in p or q.

Here the requirement on the declared variable v formalizes the concept of *locality*, i.e., that the value of v when c begins execution has no effect on this execution, and that the value of v when c finishes execution has no effect on the rest of the program.

Locality is a property of the specification of a command rather than just the command itself, since it depends on the use to which the command may be put. For example, the same command satisfies both of the specifications

$$\{\textbf{true}\}\ \mathsf{t} := \mathsf{x} + \mathsf{y}\ ;\ \mathsf{z} := \mathsf{t} \times \mathsf{t}\ \{\mathsf{z} = (\mathsf{x} + \mathsf{y})^2\}$$

$$\{\textbf{true}\}\ \mathsf{t} := \mathsf{x} + \mathsf{y}\ ;\ \mathsf{z} := \mathsf{t} \times \mathsf{t}\ \{\mathsf{z} = (\mathsf{x} + \mathsf{y})^2 \wedge \mathsf{t} = \mathsf{x} + \mathsf{y}\}.$$

But the variable t is local only in the first case, so that **newvar** t **in** $(\mathsf{t} := \mathsf{x} + \mathsf{y}\ ;\ \mathsf{z} := \mathsf{t} \times \mathsf{t})$ meets only the first specification:

$$\{\textbf{true}\}\ \textbf{newvar}\ \mathsf{t}\ \textbf{in}\ (\mathsf{t} := \mathsf{x} + \mathsf{y}\ ;\ \mathsf{z} := \mathsf{t} \times \mathsf{t})\ \{\mathsf{z} = (\mathsf{x} + \mathsf{y})^2\}.$$

Rules for Annotated Specifications:

Sequential Composition (SQ)

$$\frac{\{p\}\ c_1\ \{q\} \qquad \{q\}\ c_2\ \{r\}}{\{p\}\ c_1\ ;\{q\}\ c_2\ \{r\}}$$

Strengthening Precedent (SP)

$$\frac{p \Rightarrow q \qquad \{q\}\ c\ \{r\}}{\{p\}\{q\}\ c\ \{r\}}$$

Weakening Consequent (WC)

$$\frac{\{p\}\ c\ \{q\} \qquad q \Rightarrow r}{\{p\}\ c\ \{q\}\{r\}}$$

Rules for Annotated Specifications:

Assignment (AS)

$$\frac{p_0 \Rightarrow (p/v \rightarrow e)}{\{p_0\}\ v := e\ \{p\}} \qquad \frac{\{p_0\}\ c\ \{p/v \rightarrow e\}}{\{p_0\}\ c\ ;\ v := e\ \{p\}}$$

skip (SK)

$$\frac{p_0 \Rightarrow p}{\{p_0\}\ \mathbf{skip}\ \{p\}}$$

Partial Correctness of **while** (WHP)

$$\frac{\{i \wedge b\}\ c\ \{i\} \qquad (i \wedge \neg\, b) \Rightarrow p}{\{i\}\ \mathbf{while}\ b\ \mathbf{do}\ c\ \{p\}}$$

Total Correctness of **while** (WHT)

$$\frac{[\ i \wedge b \wedge e = v_0\ ]\ c\ [\ i \wedge e < v_0\ ] \quad (i \wedge b) \Rightarrow e \geq 0 \quad (i \wedge \neg\ b) \Rightarrow p}{[\ i\ ]\ \mathbf{while}\ [\mathbf{vrnt}\colon e]\ b\ \mathbf{do}\ c\ [\ p\ ]}$$

when v_0 does not occur free in i, b, c, or e.

The rules for conditionals and variable declarations remain unchanged.

An Annotated Specification

$$
\begin{array}{l}
[\mathsf{n} \geq 0] \\
\textbf{if } \mathsf{n} = 0 \textbf{ then } \mathsf{f} := 0 \textbf{ else} \\
\quad \textbf{newvar } \mathsf{g} \textbf{ in newvar } \mathsf{k} \textbf{ in} \\
\qquad (\mathsf{k} := 1 \,;\, \mathsf{g} := 0 \,;\, \mathsf{f} := 1 \,; \\
\qquad [\mathsf{f} = \mathrm{fib}(\mathsf{k}) \wedge \mathsf{g} = \mathrm{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n}] \\
\qquad \textbf{while } [\textbf{vrnt}\colon \mathsf{n} - \mathsf{k}]\, \mathsf{k} \neq \mathsf{n} \textbf{ do} \\
\qquad\quad (\mathsf{t} := \mathsf{g} \,;\, \mathsf{g} := \mathsf{f} \,;\, \mathsf{f} := \mathsf{f} + \mathsf{t} \,;\, \mathsf{k} := \mathsf{k} + 1)) \\
[\mathsf{f} = \mathrm{fib}(\mathsf{n})]
\end{array}
$$

Structural Inference Rules:

Renaming (RN)

$$\frac{\{p\}\ c\ \{q\}}{\{p'\}\ c'\ \{q'\},}$$

where p', c', and q' are obtained from p, c, and q by zero or more renamings of subphrases.

Free Variable Substitution (FVS)

$$\frac{\{p\}\ c\ \{q\}}{(\{p\}\ c\ \{q\})/v_1 \to e_1, \ldots, v_n \to e_n,}$$

where $v_1, \ldots, v_n$ are the variables occurring free in p, c, or q, and, if v_i is assigned by c, then e_i is a variable that does not occur free in any other e_j.

For example, in

$$\{\mathsf{x} = \mathsf{y}\}\ \mathsf{x} := \mathsf{x} + \mathsf{y}\ \{\mathsf{x} = 2 \times \mathsf{y}\},$$

one can substitute $\mathsf{x}, \mathsf{y} \to \mathsf{z}, 2 \times \mathsf{w} - 1$ to infer

$$\{\mathsf{z} = 2 \times \mathsf{w} - 1\}\ \mathsf{z} := \mathsf{z} + 2 \times \mathsf{w} - 1\ \{\mathsf{z} = 2 \times (2 \times \mathsf{w} - 1)\}.$$

But one cannot substitute $\mathsf{x}, \mathsf{y} \to \mathsf{z}, 2 \times \mathsf{z} - 1$ to infer

$$\{\mathsf{z} = 2 \times \mathsf{z} - 1\}\ \mathsf{z} := \mathsf{z} + 2 \times \mathsf{z} - 1\ \{\mathsf{z} = 2 \times (2 \times \mathsf{z} - 1)\}.$$

Free Variable Substitution (continued)

$$\frac{\{p\}\ c\ \{q\}}{(\{p\}\ c\ \{q\})/v_1 \to e_1,\ \ldots\ ,\ v_n \to e_n,}$$

where $v_1, \ldots, v_n$ are the variables occurring free in p, c, or q, and, if v_i is assigned by c, then e_i is a variable that does not occur free in any other e_j.

Proposition 7 *The free variable substitution rule is sound.*

PROOF Let δ abbreviate the substitution

$$v_1 \to e_1, \ldots, v_n \to e_n,$$

and for any store s, let

$$\widehat{s} = [\, v_1 \colon [\![e_1]\!]_{\text{exp}} s \mid \ldots \mid v_n \colon [\![e_n]\!]_{\text{exp}} s \,].$$

- Assume $\{p\}\ c\ \{q\}$ and $[\![p/\delta]\!]_{\text{assert}}\, s\, h$. By Proposition 6, $[\![p]\!]_{\text{assert}}\, \widehat{s}\, h$.
- Suppose $\langle c/\delta, (s,h)\rangle \rightsquigarrow^*$ **abort**. By Proposition 4, we would have $\langle c, (\widehat{s}, h)\rangle \rightsquigarrow^*$ **abort**, which contradicts $\{p\}\ c\ \{q\}$ and $[\![p]\!]_{\text{assert}}\, \widehat{s}\, h$.
- Suppose $\langle c/\delta, (s,h)\rangle \rightsquigarrow^* (s', h')$. By Proposition 4, we have $\langle c, (\widehat{s}, h)\rangle \rightsquigarrow^* (\widehat{s'}, h')$, so that $\{p\}\ c\ \{q\}$ and $[\![p]\!]_{\text{assert}}\, \widehat{s}\, h$ give $[\![q]\!]_{\text{assert}}\, \widehat{s'} h'$. Then, by Proposition 6, $[\![q/\delta]\!]_{\text{assert}}\, s' h'$.

END OF PROOF

Structural Inference Rules:

Disjunction of Assertions (DA)

$$\frac{\{p_0\}\ c\ \{q_0\} \qquad \{p_1\}\ c\ \{q_1\}}{\{p_0 \vee p_1\}\ c\ \{q_0 \vee q_1\}}$$

Conjunction of Assertions (CA)

$$\frac{\{p_0\}\ c\ \{q_0\} \qquad \{p_1\}\ c\ \{q_1\}}{\{p_0 \wedge p_1\}\ c\ \{q_0 \wedge q_1\}}$$

Existential Quantification of Assertions (EQ)

$$\frac{\{p\}\ c\ \{q\}}{\{\exists v.\ p\}\ c\ \{\exists v.\ q\}},$$

where v is not free in c.

Universal Quantification of Assertions (UQ)

$$\frac{\{p\}\ c\ \{q\}}{\{\forall v.\ p\}\ c\ \{\forall v.\ q\}},$$

where v is not free in c.

Structural Inference Rules:

The Unsound Rule of Constancy

$$\frac{\{p\}\ c\ \{q\}}{\{p \wedge r\}\ c\ \{q \wedge r\}}$$

when no variable occurs both free in r and assigned by c.

Counterexamples

$$\frac{\{\mathsf{x} = \mathsf{y} \wedge \mathsf{x} \mapsto 3\}\ [\mathsf{x}] := 4\ \{\mathsf{x} = \mathsf{y} \wedge \mathsf{x} \mapsto 4\}}{\{\mathsf{x} = \mathsf{y} \wedge \mathsf{x} \mapsto 3 \wedge \mathsf{y} \mapsto 3\}\ [\mathsf{x}] := 4\ \{\mathsf{x} = \mathsf{y} \wedge \mathsf{x} \mapsto 4 \wedge \mathsf{y} \mapsto 3\}}$$

$$\frac{\{\mathsf{x} \mapsto 3\}\ [\mathsf{x}] := 4\ \{\mathsf{x} \mapsto 4\}}{\{\mathsf{x} \mapsto 3 \wedge \mathsf{x} \mapsto 3\}\ [\mathsf{x}] := 4\ \{\mathsf{x} \mapsto 4 \wedge \mathsf{x} \mapsto 3\}}$$

Structural Inference Rules:

The Frame Rule (FR)

$$\frac{\{p\}\ c\ \{q\}}{\{p\ *\ r\}\ c\ \{q\ *\ r\}}$$

when no variable occurs both free in r and assigned by c.

Proposition 8 *The frame rule is sound.*

PROOF

- Assume $\{p\}\ c\ \{q\}$, and $[\![p * r]\!]_{\text{assert}} s\, h$. Then there are h_0, h_1 such that $h_0 \perp h_1$, $h = h_0 \cdot h_1$, $[\![p]\!]_{\text{assert}} s\, h_0$ and $[\![r]\!]_{\text{assert}} s\, h_1$.

- Suppose $\langle c, (s, h)\rangle \rightsquigarrow^*$ **abort**. Then, by Proposition 5, we would have $\langle c, (s, h_0)\rangle \rightsquigarrow^*$ **abort**, which contradicts $\{p\}\ c\ \{q\}$ and $[\![p]\!]_{\text{assert}} s\, h_0$.

- Suppose $\langle c, (s, h)\rangle \rightsquigarrow^* (s', h')$. Again, $\langle c, (s, h_0)\rangle \rightsquigarrow^*$ **abort** would contradict $\{p\}\ c\ \{q\}$ and $[\![p]\!]_{\text{assert}} s\, h_0$, so that, by Proposition 5, $\langle c, (s, h_0)\rangle \rightsquigarrow^* (s', h'_0)$, where $h'_0 \perp h_1$ and $h' = h'_0 \cdot h_1$. Then $\{p\}\ c\ \{q\}$ and $[\![p]\!]_{\text{assert}} s\, h_0$ imply that $[\![q]\!]_{\text{assert}} s' h'_0$.

- Since $\langle c, (s, h)\rangle \rightsquigarrow^* (s', h')$, Proposition 3 gives $s\, v = s' v$ for all v that are not assigned by c. Then, since these include the free variables of r, $[\![r]\!]_{\text{assert}} s\, h_1$ implies that $[\![r]\!]_{\text{assert}} s' h_1$. Thus $[\![q * r]\!]_{\text{assert}} s' h$.

END OF PROOF

Inference Rules for Mutation

The small form:

$$\frac{}{\{e \mapsto -\}\ [e] := e'\ \{e \mapsto e'\}}. \tag{1}$$

The large form:

$$\frac{}{\{(e \mapsto -)\ *\ r\}\ [e] := e'\ \{(e \mapsto e')\ *\ r\}}. \tag{2}$$

The WP form:

$$\frac{}{\{(e \mapsto -)\ *\ ((e \mapsto e') \mathrel{-\!\!*} p)\}\ [e] := e'\ \{p\}}. \tag{3}$$

One can derive (2) from (1) by using the frame rule; one can go in the opposite direction by taking r to be **emptyheap**.

One can derive (3) from (2) by taking r to be $(e \mapsto e') \mathrel{-\!\!*} p$ and using the law $q\ *\ (q \mathrel{-\!\!*} p) \Rightarrow p$:

$\{(e \mapsto -)\ *\ ((e \mapsto e') \mathrel{-\!\!*} p)\}$

$[e] := e'$

$\{(e \mapsto e')\ *\ ((e \mapsto e') \mathrel{-\!\!*} p)\}$

$\{p\}$.

One can go in the opposite direction by taking p to be $(e \mapsto e')\ * r$ and using the law $(p\ *\ r) \Rightarrow (p\ *\ (q \mathrel{-\!\!*} (q\ *\ r)))$:

$\{(e \mapsto -)\ *\ r\}$

$\{(e \mapsto -)\ *\ ((e \mapsto e') \mathrel{-\!\!*} ((e \mapsto e')\ *\ r))\}$

$[e] := e'$

$\{(e \mapsto e')\ *\ r\}$.

Inference Rules for Noninterfering Allocation

The small form:

$$\frac{}{\{\mathbf{emptyheap}\}\ v := \mathbf{cons}(e_0, \ldots, e_{n-1})\ \{v \mapsto e_0, \ldots, e_{n-1}\}}, \tag{4}$$

where $v \notin \mathrm{FV}(e_0, \ldots, e_{n-1})$.

The large form:

$$\frac{}{\{r\}\ v := \mathbf{cons}(e_0, \ldots, e_{n-1})\ \{(v \mapsto e_0, \ldots, e_{n-1})\ *\ r\}}, \tag{5}$$

where $v \notin \mathrm{FV}(e_0, \ldots, e_{n-1}, r)$.

One can derive (5) from (4) by using the frame rule; one can go in the opposite direction by taking r to be **emptyheap**.

Inference Rules for General Allocation

We assume that v, v', and v'' are distinct variables, and, for any phrase p, we write p' for $p/v \to v'$ and p'' for $p/v \to v''$.

The small form:

$$\frac{}{\{v = v' \wedge \mathbf{emptyheap}\}\ v := \mathbf{cons}(e_0, \ldots, e_{n-1})\ \{v \mapsto e'_0, \ldots, e'_{n-1}\}.} \quad (6)$$

The large form:

$$\frac{}{\{r\}\ v := \mathbf{cons}(e_0, \ldots, e_{n-1})\ \{\exists v'.\ (v \mapsto e'_0, \ldots, e'_{n-1})\ *\ r'\},} \quad (7)$$

where $v' \notin \mathrm{FV}(e_0, \ldots, e_{n-1}, r)$.

The WP form:

$$\frac{}{\{\forall v''.\ (v'' \mapsto e_0, \ldots, e_{n-1}) \mathrel{-\!\!*} p''\}\ v := \mathbf{cons}(e_0, \ldots, e_{n-1})\ \{p\},} \quad (8)$$

where $v'' \notin \mathrm{FV}(e_0, \ldots, e_{n-1}, p)$.

Let $\overline{e}$ abbreviate $e_0, \ldots, e_{n-1}$, and $\overline{e}'$ abbreviate $e'_0, \ldots, e'_{n-1}$. One can derive (7) from (6) by using the frame rule and (EQ):

$\{v = v' \wedge r'\}$	$\{r\}$
$\{(v = v' \wedge \mathbf{emptyheap})\ *\ r'\}$	$\{\exists v'.\ v = v' \wedge r'\}$
$v := \mathbf{cons}(\overline{e})$	$v := \mathbf{cons}(\overline{e})$
$\{(v \mapsto \overline{e}')\ *\ r'\},$	$\{\exists v'.\ (v \mapsto \overline{e}')\ *\ r'\}.$

Inference Rules for General Allocation (continued)

To derive (8) from (7), we choose $v' \notin \mathrm{FV}(e_0, \ldots, e_{n-1}, p)$, take r to be $\forall v''.\ (v'' \mapsto \overline{e}) \mathrel{-\!\!*} p''$, and use various laws about quantifiers, as well as $q \ * \ (q \mathrel{-\!\!*} p) \Rightarrow p$:

$$\begin{array}{l}
\{\forall v''.\ (v'' \mapsto \overline{e}) \mathrel{-\!\!*} p''\} \\
v := \mathbf{cons}(\overline{e}) \\
\{\exists v'.\ (v \mapsto \overline{e}') \ * \ (\forall v''.\ (v'' \mapsto \overline{e}') \mathrel{-\!\!*} p'')\} \\
\{\exists v'.\ (v \mapsto \overline{e}') \ * \ ((v \mapsto \overline{e}') \mathrel{-\!\!*} p)\} \\
\{\exists v'.\ p\} \\
\{p\}.
\end{array}$$

To go in the other direction, we choose $v'' \notin \mathrm{FV}(e_0, \ldots, e_{n-1}, r)$, take p to be $\exists v'.\ (v \mapsto \overline{e}') \ * \ r'$, and use various laws about quantifiers, as well as $r \Rightarrow (q \mathrel{-\!\!*} (q \ * \ r))$:

$$\begin{array}{l}
\{r\} \\
\{\forall v''.\ r\} \\
\{\forall v''.\ (v'' \mapsto \overline{e}) \mathrel{-\!\!*} ((v'' \mapsto \overline{e}) \ * \ r)\} \\
\{\forall v''.\ (v'' \mapsto \overline{e}) \mathrel{-\!\!*} (\exists v'.\ (v'' \mapsto \overline{e}') \ * \ r')\} \\
v := \mathbf{cons}(\overline{e}) \\
\{\exists v'.\ (v \mapsto \overline{e}') \ * \ r'\}.
\end{array}$$

Inference Rules for Disposal

The small form:

$$\frac{}{\{e \mapsto -\}\ \mathbf{dispose}\ e\ \{\mathbf{emptyheap}\}}. \tag{9}$$

The large (WP) form:

$$\frac{}{\{(e \mapsto -)\ *\ r\}\ \mathbf{dispose}\ e\ \{r\}}. \tag{10}$$

One can derive (10) from (9) by using the frame rule; one can go in the opposite direction by taking r to be **emptyheap**.

Inference Rules for Lookup

The small form:

$$\frac{}{\{(e \mapsto v'') \wedge v = v'\}\ v := [e]\ \{v = v'' \wedge (e' \mapsto v'')\}}, \tag{11}$$

where v, v', and v'' are distinct variables.

The large (WP) form:

$$\frac{}{\{\exists v''.\ (e \hookrightarrow v'') \wedge p''\}\ v := [e]\ \{p\}}, \tag{12}$$

where v'' does not occur free in p or e, and p'' is $p/v \to v''$.

To derive (12) from (11), we use the frame rule and the rule for existential quantification (EQ), and then use the implications $(e \hookrightarrow x) \wedge p \Rightarrow (e \mapsto x)\ *\ ((e \mapsto x) \mathrel{-\!\!*} p)$ and $q\ *\ (q \mathrel{-\!\!*} p) \Rightarrow p$:

$\{\exists v''.\ (e \hookrightarrow v'') \wedge p''\}$
$\{\exists v''.\ (e \mapsto v'')\ *\ ((e \mapsto v'') \mathrel{-\!\!*} p'')\}$
$\{\exists v', v''.\ ((e \mapsto v'') \wedge v = v')\ *\ ((e' \mapsto v'') \mathrel{-\!\!*} p'')\}$
$v := [e]$
$\{\exists v', v''.\ (v = v'' \wedge (e' \mapsto v''))\ *\ ((e' \mapsto v'') \mathrel{-\!\!*} p'')\}$
$\{\exists v', v''.\ v = v'' \wedge p''\}$
$\{\exists v', v''.\ p\}$
$\{p\}$.

Inference Rules for Lookup (continued)

To derive (11) from (12), we rename v'' in (12) to a fresh variable v''' that is distinct from v, v', or v'' and does not occur free in e. Then we take p to be $v = v'' \wedge e' \mapsto v''$:

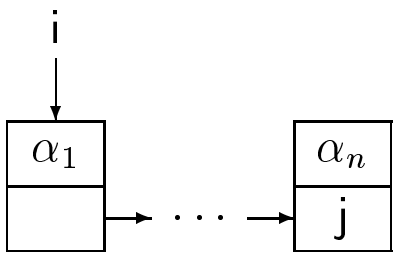
$$\{(e \mapsto v'') \wedge v = v'\}$$
$$\{(e \mapsto v'') \wedge (e' \mapsto v'')\}$$
$$\{(e \hookrightarrow v'') \wedge (e' \mapsto v'')\}$$
$$\{\exists v'''.\ (e \hookrightarrow v''') \wedge (v''' = v'' \wedge e' \mapsto v'')\}$$
$$v := [e]$$
$$\{v = v'' \wedge e' \mapsto v''\}.$$

An Annotated Specification

$$\{\mathbf{emptylist}\}$$
$$\mathsf{x} := \mathbf{cons}(\mathsf{a}, \mathsf{a})\ ;$$
$$\{\mathsf{x} \mapsto \mathsf{a}, \mathsf{a}\}$$
$$\mathsf{y} := \mathbf{cons}(\mathsf{b}, \mathsf{b})\ ;$$
$$\{\mathsf{x} \mapsto \mathsf{a}, \mathsf{a} \ * \ \mathsf{y} \mapsto \mathsf{b}, \mathsf{b}\}$$
$$[\mathsf{x} + 1] := \mathsf{y} - \mathsf{x}\ ;$$
$$\{\mathsf{x} \mapsto \mathsf{a}, \mathsf{y} - \mathsf{x} \ * \ \mathsf{y} \mapsto \mathsf{b}, \mathsf{b}\}$$
$$[\mathsf{y} + 1] := \mathsf{x} - \mathsf{y}\ ;$$
$$\{\mathsf{x} \mapsto \mathsf{a}, \mathsf{y} - \mathsf{x} \ * \ \mathsf{y} \mapsto \mathsf{b}, \mathsf{x} - \mathsf{y}\}$$
$$\{\exists \mathsf{o}.\ \mathsf{x} \mapsto \mathsf{a}, \mathsf{o} \ * \ \mathsf{y} \mapsto \mathsf{b}, -\mathsf{o}\}$$

An Inductive Definition: Singly-Linked Lists

$\mathsf{list}\ \alpha\ (\mathsf{i}, \mathsf{j})$:



Definition

$$\mathsf{list}\ \epsilon\ (\mathsf{i}, \mathsf{j}) \stackrel{\text{def}}{=} \mathbf{emptyheap} \wedge \mathsf{i} = \mathsf{j}$$

$$\mathsf{list}\ \mathsf{a}{\cdot}\alpha\ (\mathsf{i}, \mathsf{k}) \stackrel{\text{def}}{=} \exists \mathsf{j}.\ \mathsf{i} \mapsto \mathsf{a}, \mathsf{j}\ *\ \mathsf{list}\ \alpha\ (\mathsf{j}, \mathsf{k})$$

Properties

$$\mathsf{list}\ \mathsf{a}\ (\mathsf{i}, \mathsf{j}) \Leftrightarrow \mathsf{i} \mapsto \mathsf{a}, \mathsf{j}$$

$$\mathsf{list}\ \alpha{\cdot}\beta\ (\mathsf{i}, \mathsf{k}) \Leftrightarrow \exists \mathsf{j}.\ \mathsf{list}\ \alpha\ (\mathsf{i}, \mathsf{j})\ *\ \mathsf{list}\ \beta\ (\mathsf{j}, \mathsf{k})$$

$$\mathsf{list}\ \alpha{\cdot}\mathsf{b}\ (\mathsf{i}, \mathsf{k}) \Leftrightarrow \exists \mathsf{j}.\ \mathsf{list}\ \alpha\ (\mathsf{i}, \mathsf{j})\ *\ \mathsf{j} \mapsto \mathsf{b}, \mathsf{k}$$

Proof of the Composition Property

$$\mathsf{list}\ \alpha{\cdot}\beta\ (\mathsf{i},\mathsf{k}) \Leftrightarrow \exists \mathsf{j}.\ \mathsf{list}\ \alpha\ (\mathsf{i},\mathsf{j})\ *\ \mathsf{list}\ \beta\ (\mathsf{j},\mathsf{k})$$

The proof is by induction on the length of α.

When α is empty:

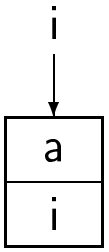
$$\begin{aligned}
&\exists \mathsf{j}.\ \mathsf{list}\ \epsilon\ (\mathsf{i},\mathsf{j})\ *\ \mathsf{list}\ \beta\ (\mathsf{j},\mathsf{k}) \\
&\quad \Leftrightarrow \exists \mathsf{j}.\ (\mathbf{emptyheap} \wedge \mathsf{i} = \mathsf{j})\ *\ \mathsf{list}\ \beta\ (\mathsf{j},\mathsf{k}) \\
&\quad \Leftrightarrow \exists \mathsf{j}.\ (\mathbf{emptyheap}\ *\ \mathsf{list}\ \beta\ (\mathsf{j},\mathsf{k})) \wedge \mathsf{i} = \mathsf{j} \\
&\quad \Leftrightarrow \exists \mathsf{j}.\ \mathsf{list}\ \beta\ (\mathsf{j},\mathsf{k}) \wedge \mathsf{i} = \mathsf{j} \\
&\quad \Leftrightarrow \mathsf{list}\ \beta\ (\mathsf{i},\mathsf{k}) \\
&\quad \Leftrightarrow \mathsf{list}\ \epsilon{\cdot}\beta\ (\mathsf{i},\mathsf{k})
\end{aligned}$$

When $\alpha = \mathsf{a}{\cdot}\alpha'$:

$$\begin{aligned}
&\exists \mathsf{j}.\ \mathsf{list}\ \mathsf{a}{\cdot}\alpha'\ (\mathsf{i},\mathsf{j})\ *\ \mathsf{list}\ \beta\ (\mathsf{j},\mathsf{k}) \\
&\quad \Leftrightarrow \exists \mathsf{j},\mathsf{l}.\ \mathsf{i} \mapsto \mathsf{a},\mathsf{l}\ *\ \mathsf{list}\ \alpha'\ (\mathsf{l},\mathsf{j})\ *\ \mathsf{list}\ \beta\ (\mathsf{j},\mathsf{k}) \\
&\quad \Leftrightarrow \exists \mathsf{l}.\ \mathsf{i} \mapsto \mathsf{a},\mathsf{l}\ *\ \mathsf{list}\ \alpha'{\cdot}\beta\ (\mathsf{l},\mathsf{k}) \qquad \text{(induction hypothesis)} \\
&\quad \Leftrightarrow \mathsf{list}\ \mathsf{a}{\cdot}\alpha'{\cdot}\beta\ (\mathsf{i},\mathsf{k})
\end{aligned}$$

Cycles

When



we have

- $\mathsf{list}\ \epsilon\ (\mathsf{i}, \mathsf{i})$ holds for the empty heap.
- $\mathsf{list}\ \mathsf{a}\ (\mathsf{i}, \mathsf{i})$ holds for the two-element heap pictured above.

But because of the $*$ in

$$\mathsf{list}\ \mathsf{a}{\cdot}\alpha\ (\mathsf{i}, \mathsf{k}) \stackrel{\text{def}}{=} \exists \mathsf{j}.\ \mathsf{i} \mapsto \mathsf{a}, \mathsf{j}\ *\ \mathsf{list}\ \alpha\ (\mathsf{j}, \mathsf{k}),$$

$\mathsf{list}\ \mathsf{a}{\cdot}\mathsf{a}\ (\mathsf{i}, \mathsf{i})$, $\mathsf{list}\ \mathsf{a}{\cdot}\mathsf{a}{\cdot}\mathsf{a}\ (\mathsf{i}, \mathsf{i})$, ... do not hold.

Emptyness Conditions

$$\mathsf{list}\ \alpha\ (\mathsf{i}, \mathsf{j}) \Rightarrow \left(\begin{array}{ll} & (\mathsf{i} = \mathbf{nil} \Rightarrow (\alpha = \epsilon \wedge \mathsf{j} = \mathbf{nil})) \\ \wedge & (\mathsf{i} \neq \mathsf{j} \Rightarrow \alpha \neq \epsilon). \end{array} \right)$$

Inserting a List Element

- At the beginning:

$$
\begin{array}{l}
\{\mathbf{list}\ \alpha\ (\mathsf{i},\mathsf{j})\} \\
\mathsf{k} := \mathbf{cons}(\mathsf{a},\mathsf{i}); \\
\{\mathbf{list}\ \alpha\ (\mathsf{i},\mathsf{j})\ *\ \mathsf{k} \mapsto \mathsf{a},\mathsf{i}\} \\
\{\exists \mathsf{i}.\ \mathsf{k} \mapsto \mathsf{a},\mathsf{i}\ *\ \mathbf{list}\ \alpha\ (\mathsf{i},\mathsf{j})\} \\
\{\mathbf{list}\ \mathsf{a}{\cdot}\alpha\ (\mathsf{k},\mathsf{j})\} \\
\mathsf{i} := \mathsf{k} \\
\{\mathbf{list}\ \mathsf{a}{\cdot}\alpha\ (\mathsf{i},\mathsf{j})\}
\end{array}
$$

- At the end of a nonempty segment:

$$
\begin{array}{l}
\{\mathbf{list}\ \alpha\ (\mathsf{i},\mathsf{j})\ *\ \mathsf{j} \mapsto \mathsf{a},\mathsf{k}\} \\
\mathsf{l} := \mathbf{cons}(\mathsf{b},\mathsf{k})\ ; \\
\{\mathbf{list}\ \alpha\ (\mathsf{i},\mathsf{j})\ *\ \mathsf{j} \mapsto \mathsf{a},\mathsf{k}\ *\ \mathsf{l} \mapsto \mathsf{b},\mathsf{k}\} \\
\{\mathbf{list}\ \alpha\ (\mathsf{i},\mathsf{j})\ *\ \mathsf{j} \mapsto \mathsf{a}\ *\ \mathsf{j}+1 \mapsto \mathsf{k}\ *\ \mathsf{l} \mapsto \mathsf{b},\mathsf{k}\} \\
\{\mathbf{list}\ \alpha\ (\mathsf{i},\mathsf{j})\ *\ \mathsf{j} \mapsto \mathsf{a}\ *\ \mathsf{j}+1 \mapsto -\ *\ \mathsf{l} \mapsto \mathsf{b},\mathsf{k}\} \\
[\mathsf{j}+1] := \mathsf{l} \\
\{\mathbf{list}\ \alpha\ (\mathsf{i},\mathsf{j})\ *\ \mathsf{j} \mapsto \mathsf{a}\ *\ \mathsf{j}+1 \mapsto \mathsf{l}\ *\ \mathsf{l} \mapsto \mathsf{b},\mathsf{k}\} \\
\{\mathbf{list}\ \alpha\ (\mathsf{i},\mathsf{j})\ *\ \mathsf{j} \mapsto \mathsf{a},\mathsf{l}\ *\ \mathsf{l} \mapsto \mathsf{b},\mathsf{k}\} \\
\{\mathbf{list}\ \alpha{\cdot}\mathsf{a}\ (\mathsf{i},\mathsf{l})\ *\ \mathsf{l} \mapsto \mathsf{b},\mathsf{k}\} \\
\{\mathbf{list}\ \alpha{\cdot}\mathsf{a}{\cdot}\mathsf{b}\ (\mathsf{i},\mathsf{k})\}
\end{array}
$$

Deleting a List Element

- At the beginning:

$\{\mathbf{list}\ \mathsf{a}{\cdot}\alpha\ (\mathsf{i},\mathsf{k})\}$
$\{\exists \mathsf{j}.\ \mathsf{i} \mapsto \mathsf{a},\mathsf{j}\ *\ \mathbf{list}\ \alpha\ (\mathsf{j},\mathsf{k})\}$
$\{\exists \mathsf{j}.\ (\mathsf{i} \mapsto \mathsf{a},\mathsf{j}\ *\ \mathbf{list}\ \alpha\ (\mathsf{j},\mathsf{k})) \wedge (\mathsf{i} \mapsto \mathsf{a},\mathsf{j}\ *\ \mathbf{list}\ \alpha\ (\mathsf{j},\mathsf{k}))\}$
$\{\exists \mathsf{j}.\ (\mathsf{i} \mapsto \mathsf{a},\mathsf{j}\ *\ \mathbf{list}\ \alpha\ (\mathsf{j},\mathsf{k})) \wedge (\mathsf{i} \mapsto \mathsf{a},\mathsf{j}\ *\ \mathbf{true})\}$
$\{\exists \mathsf{j}.\ (\mathsf{i} \mapsto \mathsf{a},\mathsf{j}\ *\ \mathbf{list}\ \alpha\ (\mathsf{j},\mathsf{k})) \wedge (\mathsf{i} \mapsto \mathsf{a}\ *\ \mathsf{i}+1 \mapsto \mathsf{j}\ *\ \mathbf{true})\}$
$\{\exists \mathsf{j}.\ (\mathsf{i} \mapsto \mathsf{a},\mathsf{j}\ *\ \mathbf{list}\ \alpha\ (\mathsf{j},\mathsf{k})) \wedge (\mathbf{true}\ *\ \mathsf{i}+1 \mapsto \mathsf{j}\ *\ \mathbf{true})\}$
$\{\exists \mathsf{j}.\ (\mathsf{i} \mapsto \mathsf{a},\mathsf{j}\ *\ \mathbf{list}\ \alpha\ (\mathsf{j},\mathsf{k})) \wedge (\mathsf{i}+1 \mapsto \mathsf{j}\ *\ \mathbf{true})\}$
$\{\exists \mathsf{j}.\ (\mathsf{i} \mapsto \mathsf{a},\mathsf{j}\ *\ \mathbf{list}\ \alpha\ (\mathsf{j},\mathsf{k})) \wedge \mathsf{i}+1 \hookrightarrow \mathsf{j}\}$
$\mathsf{j} := [\mathsf{i}+1]\ ;$
$\{\mathsf{i} \mapsto \mathsf{a},\mathsf{j}\ *\ \mathbf{list}\ \alpha\ (\mathsf{j},\mathsf{k})\}$
$\{\mathsf{i} \mapsto \mathsf{a}\ *\ \mathsf{i}+1 \mapsto \mathsf{j}\ *\ \mathbf{list}\ \alpha\ (\mathsf{j},\mathsf{k})\}$
$\mathbf{dispose}\ \mathsf{i}\ ;$
$\{\mathsf{i}+1 \mapsto \mathsf{j}\ *\ \mathbf{list}\ \alpha\ (\mathsf{j},\mathsf{k})\}$
$\mathbf{dispose}\ \mathsf{i}+1\ ;$
$\{\mathbf{list}\ \alpha\ (\mathsf{j},\mathsf{k})\}$
$\mathsf{i} := \mathsf{j}$
$\{\mathbf{list}\ \alpha\ (\mathsf{i},\mathsf{k})\}$

Deleting a List Element (continued)

- At the end of a nonempty segment:

$$\{\mathbf{list}\ \alpha\,(\mathsf{i},\mathsf{j})\ *\ \mathsf{j}\mapsto\mathsf{a},\mathsf{k}\ *\ \mathsf{k}\mapsto\mathsf{b},\mathsf{l}\}$$
$$[\mathsf{j}+1]:=\mathsf{l}\,;$$
$$\{\mathbf{list}\ \alpha\,(\mathsf{i},\mathsf{j})\ *\ \mathsf{j}\mapsto\mathsf{a},\mathsf{l}\ *\ \mathsf{k}\mapsto\mathsf{b},\mathsf{l}\}$$
$$\mathbf{dispose}\ \mathsf{k}\,;\ \mathbf{dispose}\ \mathsf{k}+1$$
$$\{\mathbf{list}\ \alpha\,(\mathsf{i},\mathsf{j})\ *\ \mathsf{j}\mapsto\mathsf{a},\mathsf{l}\}$$
$$\{\mathbf{list}\ \alpha{\cdot}\mathsf{a}\,(\mathsf{i},\mathsf{l})\}$$

Reversing a List

$\{\mathbf{list}\ \gamma\ (\mathsf{i}, \mathbf{nil})\}$

$\{\mathbf{list}\ \gamma\ (\mathsf{i}, \mathbf{nil})\ *\ \mathbf{emptyheap}\}$

$\mathsf{j} := \mathbf{nil}\ ;$

$\{\mathbf{list}\ \gamma\ (\mathsf{i}, \mathbf{nil})\ *\ (\mathbf{emptyheap} \wedge \mathsf{j} = \mathbf{nil})\}$

$\{\mathbf{list}\ \gamma\ (\mathsf{i}, \mathbf{nil})\ *\ \mathbf{list}\ \epsilon\ (\mathsf{j}, \mathbf{nil})\}$

$\{\exists \alpha, \beta.\ (\mathbf{list}\ \alpha\ (\mathsf{i}, \mathbf{nil})\ *\ \mathbf{list}\ \beta\ (\mathsf{j}, \mathbf{nil})) \wedge \gamma^\dagger = \alpha^\dagger \cdot \beta\}$

$\mathbf{while}\ \mathsf{i} \neq \mathbf{nil}\ \mathbf{do}$

$\big($

$\{\exists \mathsf{a}, \alpha, \beta.\ (\mathbf{list}\ \mathsf{a}\cdot\alpha\ (\mathsf{i}, \mathbf{nil})\ *\ \mathbf{list}\ \beta\ (\mathsf{j}, \mathbf{nil})) \wedge \gamma^\dagger = (\mathsf{a}\cdot\alpha)^\dagger \cdot \beta\}$

$\{\exists \mathsf{a}, \alpha, \beta, \mathsf{k}.\ (\mathsf{i} \mapsto \mathsf{a}, \mathsf{k}\ *\ \mathbf{list}\ \alpha\ (\mathsf{k}, \mathbf{nil})\ *\ \mathbf{list}\ \beta\ (\mathsf{j}, \mathbf{nil}))$
$\wedge\ \gamma^\dagger = (\mathsf{a}\cdot\alpha)^\dagger \cdot \beta\}$

$\mathsf{k} := [\mathsf{i} + 1]\ ;$

$\{\exists \mathsf{a}, \alpha, \beta.\ (\mathsf{i} \mapsto \mathsf{a}, \mathsf{k}\ *\ \mathbf{list}\ \alpha\ (\mathsf{k}, \mathbf{nil})\ *\ \mathbf{list}\ \beta\ (\mathsf{j}, \mathbf{nil}))$
$\wedge\ \gamma^\dagger = (\mathsf{a}\cdot\alpha)^\dagger \cdot \beta\}$

$[\mathsf{i} + 1] := \mathsf{j}\ ;$

$\{\exists \mathsf{a}, \alpha, \beta.\ (\mathsf{i} \mapsto \mathsf{a}, \mathsf{j}\ *\ \mathbf{list}\ \alpha\ (\mathsf{k}, \mathbf{nil})\ *\ \mathbf{list}\ \beta\ (\mathsf{j}, \mathbf{nil}))$
$\wedge\ \gamma^\dagger = (\mathsf{a}\cdot\alpha)^\dagger \cdot \beta\}$

$\{\exists \mathsf{a}, \alpha, \beta.\ (\mathbf{list}\ \alpha\ (\mathsf{k}, \mathbf{nil})\ *\ \mathbf{list}\ \mathsf{a}\cdot\beta\ (\mathsf{i}, \mathbf{nil})) \wedge \gamma^\dagger = \alpha^\dagger \cdot \mathsf{a} \cdot \beta\}$

$\{\exists \alpha, \beta.\ (\mathbf{list}\ \alpha\ (\mathsf{k}, \mathbf{nil})\ *\ \mathbf{list}\ \beta\ (\mathsf{i}, \mathbf{nil})) \wedge \gamma^\dagger = \alpha^\dagger \cdot \beta\}$

$\mathsf{j} := \mathsf{i}\ ;\ \mathsf{i} := \mathsf{k}$

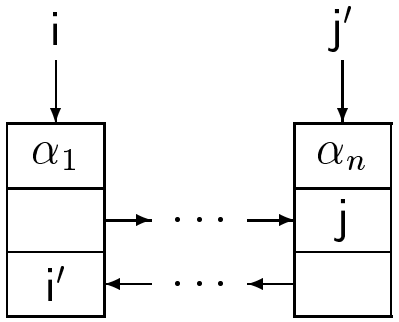
$\{\exists \alpha, \beta.\ (\mathbf{list}\ \alpha\ (\mathsf{i}, \mathbf{nil})\ *\ \mathbf{list}\ \beta\ (\mathsf{j}, \mathbf{nil})) \wedge \gamma^\dagger = \alpha^\dagger \cdot \beta\}\big)$

$\{\exists \alpha, \beta.\ \mathbf{list}\ \beta\ (\mathsf{j}, \mathbf{nil}) \wedge \gamma^\dagger = \alpha^\dagger \cdot \beta \wedge \alpha = \epsilon\}$

$\{\mathbf{list}\ \gamma^\dagger\ (\mathsf{j}, \mathbf{nil})\}$

An Inductive Definition: Doubly-Linked Lists

$\mathsf{dlist}\ \alpha\ (\mathsf{i}, \mathsf{i}', \mathsf{j}, \mathsf{j}')$:



Definition

$$\mathsf{dlist}\ \epsilon\ (\mathsf{i}, \mathsf{i}', \mathsf{j}, \mathsf{j}') \stackrel{\text{def}}{=} \mathbf{emptyheap} \wedge \mathsf{i} = \mathsf{j} \wedge \mathsf{i}' = \mathsf{j}'$$

$$\mathsf{dlist}\ \mathsf{a}{\cdot}\alpha\ (\mathsf{i}, \mathsf{i}', \mathsf{k}, \mathsf{k}') \stackrel{\text{def}}{=} \exists \mathsf{j}.\ \mathsf{i} \mapsto \mathsf{a}, \mathsf{j}, \mathsf{i}' \ *\ \mathsf{dlist}\ \alpha\ (\mathsf{j}, \mathsf{i}, \mathsf{k}, \mathsf{k}').$$

Properties

$$\mathsf{dlist}\ \mathsf{a}\ (\mathsf{i}, \mathsf{i}', \mathsf{j}, \mathsf{j}') \Leftrightarrow \mathsf{i} \mapsto \mathsf{a}, \mathsf{j}, \mathsf{i}' \wedge \mathsf{i} = \mathsf{j}'$$

$$\mathsf{dlist}\ \alpha{\cdot}\beta\ (\mathsf{i}, \mathsf{i}', \mathsf{k}, \mathsf{k}') \Leftrightarrow \exists \mathsf{j}, \mathsf{j}'.\ \mathsf{dlist}\ \alpha\ (\mathsf{i}, \mathsf{i}', \mathsf{j}, \mathsf{j}') \ *\ \mathsf{dlist}\ \beta\ (\mathsf{j}, \mathsf{j}', \mathsf{k}, \mathsf{k}')$$

$$\mathsf{dlist}\ \alpha{\cdot}\mathsf{b}\ (\mathsf{i}, \mathsf{i}', \mathsf{k}, \mathsf{k}') \Leftrightarrow \exists \mathsf{j}'.\ \mathsf{dlist}\ \alpha\ (\mathsf{i}, \mathsf{i}', \mathsf{k}', \mathsf{j}') \ *\ \mathsf{k}' \mapsto \mathsf{b}, \mathsf{k}, \mathsf{j}'.$$

Emptyness Conditions

$$\mathsf{dlist}\ \alpha\ (\mathsf{i}, \mathsf{i}', \mathsf{j}, \mathsf{j}') \Rightarrow \left(\begin{array}{rl} & (\mathsf{i} = \mathbf{nil} \Rightarrow (\alpha = \epsilon \wedge \mathsf{j} = \mathbf{nil} \wedge \mathsf{i}' = \mathsf{j}')) \\ \wedge & (\mathsf{j}' = \mathbf{nil} \Rightarrow (\alpha = \epsilon \wedge \mathsf{i}' = \mathbf{nil} \wedge \mathsf{i} = \mathsf{j})) \\ \wedge & (\mathsf{i} \neq \mathsf{j} \Rightarrow \alpha \neq \epsilon) \\ \wedge & (\mathsf{i}' \neq \mathsf{j}' \Rightarrow \alpha \neq \epsilon) \end{array} \right)$$

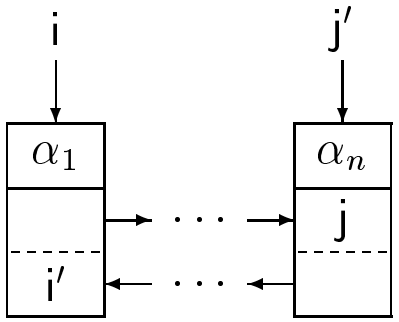
Xor-Linked Lists

$$\mathsf{xlist}\ \epsilon\ (\mathsf{i}, \mathsf{i}', \mathsf{j}, \mathsf{j}') \stackrel{\text{def}}{=} \mathbf{emptyheap} \wedge \mathsf{i} = \mathsf{j} \wedge \mathsf{i}' = \mathsf{j}'$$

$$\mathsf{xlist}\ \mathsf{a}{\cdot}\alpha\ (\mathsf{i}, \mathsf{i}', \mathsf{k}, \mathsf{k}') \stackrel{\text{def}}{=} \exists \mathsf{j}.\ \mathsf{i} \mapsto \mathsf{a}, (\mathsf{j} \oplus \mathsf{i}') \ *\ \mathsf{xlist}\ \alpha\ (\mathsf{j}, \mathsf{i}, \mathsf{k}, \mathsf{k}').$$

Xor-Linked Lists

$\mathsf{xlist}\ \alpha\ (\mathsf{i}, \mathsf{i}', \mathsf{j}, \mathsf{j}')$:



$$\mathsf{xlist}\ \epsilon\ (\mathsf{i}, \mathsf{i}', \mathsf{j}, \mathsf{j}') \stackrel{\text{def}}{=} \mathbf{emptyheap} \wedge \mathsf{i} = \mathsf{j} \wedge \mathsf{i}' = \mathsf{j}'$$

$$\mathsf{xlist}\ \mathsf{a}{\cdot}\alpha\ (\mathsf{i}, \mathsf{i}', \mathsf{k}, \mathsf{k}') \stackrel{\text{def}}{=} \exists \mathsf{j}.\ \mathsf{i} \mapsto \mathsf{a}, (\mathsf{j} \oplus \mathsf{i}') \ * \ \mathsf{xlist}\ \alpha\ (\mathsf{j}, \mathsf{i}, \mathsf{k}, \mathsf{k}').$$

Properties

$$\mathsf{xlist}\ \mathsf{a}\ (\mathsf{i}, \mathsf{i}', \mathsf{j}, \mathsf{j}') \Leftrightarrow \mathsf{i} \mapsto \mathsf{a}, (\mathsf{j} \oplus \mathsf{i}') \wedge \mathsf{i} = \mathsf{j}'$$

$$\mathsf{xlist}\ \alpha{\cdot}\beta\ (\mathsf{i}, \mathsf{i}', \mathsf{k}, \mathsf{k}') \Leftrightarrow \exists \mathsf{j}, \mathsf{j}'.\ \mathsf{xlist}\ \alpha\ (\mathsf{i}, \mathsf{i}', \mathsf{j}, \mathsf{j}') \ * \ \mathsf{xlist}\ \beta\ (\mathsf{j}, \mathsf{j}', \mathsf{k}, \mathsf{k}')$$

$$\mathsf{xlist}\ \alpha{\cdot}\mathsf{b}\ (\mathsf{i}, \mathsf{i}', \mathsf{k}, \mathsf{k}') \Leftrightarrow \exists \mathsf{j}'.\ \mathsf{xlist}\ \alpha\ (\mathsf{i}, \mathsf{i}', \mathsf{k}', \mathsf{j}') \ * \ \mathsf{k}' \mapsto \mathsf{b}, (\mathsf{k} \oplus \mathsf{j}').$$

Emptyness Conditions

$$\mathsf{xlist}\ \alpha\ (\mathsf{i}, \mathsf{i}', \mathsf{j}, \mathsf{j}') \Rightarrow \left(\begin{array}{rr} & (\mathsf{i} = \mathbf{nil} \Rightarrow (\alpha = \epsilon \wedge \mathsf{j} = \mathbf{nil} \wedge \mathsf{i}' = \mathsf{j}')) \\ \wedge & (\mathsf{j}' = \mathbf{nil} \Rightarrow (\alpha = \epsilon \wedge \mathsf{i}' = \mathbf{nil} \wedge \mathsf{i} = \mathsf{j})) \\ \wedge & (\mathsf{i} \neq \mathsf{j} \Rightarrow \alpha \neq \epsilon) \\ \wedge & (\mathsf{i}' \neq \mathsf{j}' \Rightarrow \alpha \neq \epsilon) \end{array} \right)$$