Lectures on Reasoning about Shared Mutable Data Structure

John C. Reynolds
Department of Computer Science
Carnegie Mellon University

Tandil, Argentina

September 2000

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] (who 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; it is this extension that will be used in these lectures.

We will begin with the denotational semantics of predicate logic and a transition-based operational semantics of the simple imperative language, whose marriage will give traditional Hoare logic. (A more detailed presentation can be found in [17].)

Then we will extend the programming language with new commands for dealing with shared mutable structure, extend predicate calculus with new assertions and operations for describing such structure, and add inference rules to Hoare logic to support reasoning about these extensions.

Finally, we will use inductive definitions for predicates on abstract data types to define singly and doubly-linked lists, which be used in a variety of examples.

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

The diagrams on pages 68 and 72 are used with the kind permission of Peter O'Hearn.

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 *Millennial Perspectives in Computer Science*. Palgrave, 2000.

[21] Samin Ishtiaq and Peter W. O'Hearn. BI as an assertion language for mutable data structure. To appear in POPL 2001, March 10, 2000.

[22] David Walker and Greg Morrisett. Alias types for recursive data structures. In *Workshop on Types in Compilation*, 2000.

An Abstract Grammar for Predicate Logic

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

$$\langle \text{assert} \rangle ::= \textbf{true} \mid \textbf{false}$$
$$\mid \langle \text{exp} \rangle = \langle \text{exp} \rangle \mid \langle \text{exp} \rangle \neq \langle \text{exp} \rangle \mid \cdots$$
$$\mid \neg \langle \text{assert} \rangle \mid \langle \text{assert} \rangle \wedge \langle \text{assert} \rangle \mid \langle \text{assert} \rangle \vee \langle \text{assert} \rangle$$
$$\mid \langle \text{assert} \rangle \Rightarrow \langle \text{assert} \rangle \mid \langle \text{assert} \rangle \Leftrightarrow \langle \text{assert} \rangle$$
$$\mid \forall \langle \text{var} \rangle .\ \langle \text{assert} \rangle \mid \exists \langle \text{var} \rangle .\ \langle \text{assert} \rangle$$

Concrete Representation

- In decreasing order of precedence:

  $\times \; (-_{\text{unary}} + -_{\text{binary}}) \; (= \neq \mapsto \hookrightarrow) \; \neg \; (\wedge \;\; *) \vee (\Rightarrow \;\; -\!\!*) \;\; \Leftrightarrow \;\; := \; ;$

- Each of the binary operators is left associative.

- A quantified expression $\forall v.\ a$ or $\exists v.\ a$ may follow any boolean operator, and will extend to the first *stopping symbol*, i.e.

  - an unmatched closing delimiter:

    $)\quad]\quad \}\quad :\quad |\quad$ **do** **else** **in** **then**

  - or, when unparenthesized by any delimiters:

    $;\quad ,$

  or to the end of the enclosing phrase.

For example,

$(\mathsf{y} + 1 = \mathsf{z} \wedge \forall \mathsf{x}.\ \mathsf{x} = 0 \vee \mathsf{x} \neq 0) \Leftrightarrow \mathsf{y} = \mathsf{z} - 1$ is

$([[\mathsf{y} + 1] = \mathsf{z}] \wedge [\forall \mathsf{x}.\ [[\mathsf{x} = 0] \vee [\mathsf{x} \neq 0]]]) \Leftrightarrow [\mathsf{y} = [\mathsf{z} - 1]].$

The Denotational Semantics of Predicate Logic

Let States be the set of states

$$\text{States} = \langle \text{var} \rangle \to \mathbf{Z},$$

where

$$\mathbf{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\} \qquad \mathbf{B} = \{\mathbf{true}, \mathbf{false}\}.$$

Then the *semantic functions*

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

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

are defined by *semantic equations.*

The Semantic Equations

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

$$[\![v]\!]_{\mathrm{exp}}\sigma = \sigma v,$$

$$[\![-e]\!]_{\mathrm{exp}}\sigma = -[\![e]\!]_{\mathrm{exp}}\sigma,$$

$$[\![e_0 + e_1]\!]_{\mathrm{exp}}\sigma = [\![e_0]\!]_{\mathrm{exp}}\sigma + [\![e_1]\!]_{\mathrm{exp}}\sigma$$

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

$$[\![\mathbf{true}]\!]_{\mathrm{assert}}\sigma = \mathbf{true},$$

$$[\![\mathbf{false}]\!]_{\mathrm{assert}}\sigma = \mathbf{false},$$

$$[\![e_0 = e_1]\!]_{\mathrm{assert}}\sigma = ([\![e_0]\!]_{\mathrm{exp}}\sigma = [\![e_1]\!]_{\mathrm{exp}}\sigma)$$

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

$$[\![\neg p]\!]_{\mathrm{assert}}\sigma = \neg[\![p]\!]_{\mathrm{assert}}\sigma,$$

$$[\![p_0 \wedge p_1]\!]_{\mathrm{assert}}\sigma = [\![p_0]\!]_{\mathrm{assert}}\sigma \wedge [\![p_1]\!]_{\mathrm{assert}}\sigma$$

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

$$[\![\forall v.\ p]\!]_{\mathrm{assert}}\sigma = \forall n \in \mathbf{Z}.\ [\![p]\!]_{\mathrm{assert}}[\,\sigma \mid v{:}\,n\,],$$

$$[\![\exists v.\ p]\!]_{\mathrm{assert}}\sigma = \exists n \in \mathbf{Z}.\ [\![p]\!]_{\mathrm{assert}}[\,\sigma \mid v{:}\,n\,].$$

An Example

Suppose $\sigma\,\mathsf{x} = 3$. Then:

$$\begin{aligned}
[\![\mathsf{x} + 0 = \mathsf{x}]\!]_{\text{assert}}\sigma &= ([\![\mathsf{x} + 0]\!]_{\text{exp}}\sigma = [\![\mathsf{x}]\!]_{\text{exp}}\sigma) \\
&= ([\![\mathsf{x}]\!]_{\text{exp}}\sigma + [\![0]\!]_{\text{exp}}\sigma = [\![\mathsf{x}]\!]_{\text{exp}}\sigma) \\
&= (3 + 0 = 3) \\
&= \mathbf{true}.
\end{aligned}$$

Moreover, the above argument does not depend on the choice of 3; it holds for any integer n that x might be mapped into by σ. Thus, for any state σ,

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

Some Terminology

- When $[\![p]\!]_{\text{assert}}\sigma = \mathbf{true}$ we say
  - p is *true* in σ,
  - p *holds* for σ,
  - p *describes* σ,
  - σ *satisfies* p.
- When $[\![p]\!]_{\text{assert}}\sigma = \mathbf{true}$ for all states $\sigma \in$ States, we say
  - p is *valid*.
- When $[\![p]\!]_{\text{assert}}\sigma = \mathbf{false}$ for all states $\sigma \in$ States, we say
  - p is *unsatisfiable*.
- When $[\![q]\!]_{\text{assert}}\sigma = \mathbf{true}$ holds for every state $\sigma \in$ States such that $[\![p]\!]_{\text{assert}}\sigma = \mathbf{true}$, 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 $[\![q]\!]_{\text{assert}}\sigma = \mathbf{true}$ holds for exactly those states $\sigma \in$ States such that $[\![p]\!]_{\text{assert}}\sigma = \mathbf{true}$, we say
  - p is *equivalent* to q.

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 later, 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)

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

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.

Binding

- Certain occurrences of variables are *binders*, each of which has an associated phrase called its *scope*.
  - In predicate logic, the first occurrence of v in $\forall v.\ p$ or $\exists v.\ p$ is a binder, and its scope is p.
  - In a programming language with local variable declarations, the first occurrences of v in **newvar** $v := e$ **in** c is a binder and its scope is c.
- If an occurrence of v (which is not itself a binder) lies within the scope of a binder of v, it is said to be *bound* by the binder; otherwise, it is a *free* occurrence.
- When the occurrence of v lies within the scopes of several binders of v, it is bound by the one with the smallest scope.
- We write $\mathrm{FV}_{\mathrm{exp}}(p)$ or $\mathrm{FV}_{\mathrm{assert}}(p)$ for the set of variables that occur free in p.

For example:

$$\begin{array}{ccccccccccccccccc} \forall & \mathsf{x} & . & (& \mathsf{x} & \neq & \mathsf{y} & \vee & \forall & \mathsf{y} & . & (& \mathsf{x} & = & \mathsf{y} & \vee & \forall & \mathsf{x} & . & \mathsf{x} & + & \mathsf{y} & \neq & \mathsf{x} &) &) \\ & 1 & & & 1 & & & & & 2 & & & 1 & & 2 & & & 3 & & 3 & & 2 & & 3 & & & \end{array}$$

$$\mathrm{FV}_{\mathrm{assert}}(\forall \mathsf{x}.\ \mathsf{x} + \mathsf{y} \neq \mathsf{x}) = \{\mathsf{y}\}$$

$$\mathrm{FV}_{\mathrm{assert}}(\mathsf{x} = \mathsf{y} \vee \forall \mathsf{x}.\ \mathsf{x} + \mathsf{y} \neq \mathsf{x}) = \{\mathsf{x}, \mathsf{y}\}$$

Renaming

- If $v_{\text{new}} \notin \text{FV}_{\text{assert}}(p) - \{v\}$, then

$$\forall v_{\text{new}}.\ (p/v \to v_{\text{new}})$$

  is said to be a *renaming* of

$$\forall v.\ p.$$

- If $v_{\text{new}} \notin \text{FV}_{\text{comm}}(c) - \{v\}$, then

$$\textbf{newvar}\ v_{\text{new}} := e\ \textbf{in}\ (c/v \to v_{\text{new}})$$

  is said to be a *renaming* of

$$\textbf{newvar}\ v := e\ \textbf{in}\ c.$$

For all languages considered in these lectures, renaming preserves meaning.

Substitution

Consider the sound inference rule

$$\frac{}{(\forall v.\ p) \Rightarrow (p/v \rightarrow e)}\,,$$

where $p/v \rightarrow e$ denotes the result of substituting e for v in p. Suppose we replace

$$\begin{array}{lll} p & \text{by} & \exists \mathsf{y}.\ \mathsf{y} > \mathsf{x} \\ v & \text{by} & \mathsf{x} \\ e & \text{by} & \mathsf{y} + 1 \end{array}$$

to obtain the instance

$$(\forall \mathsf{x}.\ \exists \mathsf{y}.\ \mathsf{y} > \mathsf{x}) \Rightarrow \Big((\exists \mathsf{y}.\ \mathsf{y} > \mathsf{x})/\mathsf{x} \rightarrow \mathsf{y} + 1\Big).$$

If the substitution $\mathsf{x} \rightarrow \mathsf{y} + 1$ merely replaces x by $\mathsf{y} + 1$, we get

$$(\forall \mathsf{x}.\ \exists \mathsf{y}.\ \mathsf{y} > \mathsf{x}) \Rightarrow (\exists \mathsf{y}.\ \mathsf{y} > \mathsf{y} + 1),$$

which is clearly false. Instead, before we carry out the substitution, we must rename the binder y in the second occurrence of $\exists \mathsf{y}.\ \mathsf{y} > \mathsf{x}$ so that it does not *capture* the occurrence of y in $\mathsf{y}+1$:

$$(\forall \mathsf{x}.\ \exists \mathsf{y}.\ \mathsf{y} > \mathsf{x}) \Rightarrow \Big((\exists \mathsf{z}.\ \mathsf{z} > \mathsf{x})/\mathsf{x} \rightarrow \mathsf{y} + 1\Big).$$

Then the substitution gives the correct result

$$(\forall \mathsf{x}.\ \exists \mathsf{y}.\ \mathsf{y} > \mathsf{x}) \Rightarrow (\exists \mathsf{z}.\ \mathsf{z} > \mathsf{y} + 1).$$

In general, before carrying out the substitution $p/v \rightarrow e$ one must rename subphrases of p so that no variable occurring free in e occurs as a binder in p (whose scope encloses a free occurrence of v).

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

$$e = e$$

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

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

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

Exercise 1.1

State the following in predicate logic:

1. There is at least one integer larger than zero and smaller than two.

2. There is at most one integer larger than zero and smaller than two.

3. There are at least two distinct integers larger than zero and smaller than three.

4. There are at most two distinct integers larger than zero and smaller than three.

State the following in predicate logic. Assume that variables and expressions only range over natural numbers (that is, non-negative integers). Do not use the operations $\div$ or **rem**.

1. a is a divisor of b (or, equally well, b is a multiple of a).

2. a is a common divisor of b and c.

3. a is a greatest common divisor of b and c.

4. p is prime.

Exercise 1.2

Suppose that, when v is a variable and e_0, e_1, and e_2 are integer expressions,

$$\textbf{sum}\ v\colon e_0\ \textbf{to}\ e_1.\ e_2$$

is an integer expression (called a *summation* expression) with the same meaning as the conventional mathematical expression

$$\sum_{v=e_0}^{e_1} e_2.$$

Describe this extension of predicate logic by giving:

1. an abstract-grammar production;
2. a semantic equation;
3. a definition of any binders and their scopes;
4. some nontrivial inference rules.

The Simple Imperative Language: Abstract Syntax

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

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

$\langle \text{comm} \rangle ::= \langle \text{var} \rangle := \langle \text{exp} \rangle \mid \textbf{skip} \mid \langle \text{comm} \rangle \,;\, \langle \text{comm} \rangle$
$\quad \mid \textbf{if } \langle \text{boolexp} \rangle \textbf{ then } \langle \text{comm} \rangle \textbf{ else } \langle \text{comm} \rangle$
$\quad \mid \textbf{while } \langle \text{boolexp} \rangle \textbf{ do } \langle \text{comm} \rangle$
$\quad \mid \textbf{newvar } \langle \text{var} \rangle := \langle \text{exp} \rangle \textbf{ in } \langle \text{comm} \rangle$

Structural Operational Semantics

An execution (of a command) is a sequence

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

where each *configuration* γ_i is either of:

- A state (a *terminal* configuration)
- A state paired with a command (a *nonterminal* configuration)

Example:

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

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

$$\langle \mathsf{x} := \mathsf{x} + 1, [\,\mathsf{x}\colon 4 \mid \mathsf{y}\colon 11\,]\rangle \rightsquigarrow$$

$$[\,\mathsf{x}\colon 5 \mid \mathsf{y}\colon 11\,].$$

Configurations and the Transition Relation

In general, we have

- $\Gamma_N = \langle \text{comm} \rangle \times \text{States}$: the set of *nonterminal* configurations,
- $\Gamma_T = \text{States}$: the set of *terminal* configurations,
- $\Gamma = \Gamma_N \cup \Gamma_T$: the set of all configurations,
- $\rightsquigarrow$: the *transition* relation between Γ_N and Γ,
- *executions*, which are finite or infinite sequences of configurations, each of which is related to the next by the transition relation,
- the transitive and reflexive closure, $\gamma \rightsquigarrow^* \gamma'$, which holds if there is a finite execution beginning with γ and ending with γ',
- $\gamma \uparrow$, which holds if γ *diverges*, i.e. if there is an infinite execution beginning with γ.

The transition relation will be defined by inference rules.

Totality

- 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.
- For every configuration γ, there is at least one execution sequence beginning with γ that is either infinite or ends with a terminal configuration.

Transition Rules for the Simple Imperative Language

- Assignment

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

- **skip**

$$\frac{}{\langle \mathbf{skip}, \sigma\rangle \rightsquigarrow \sigma}$$

- Sequencing

$$\frac{\langle c_0, \sigma\rangle \rightsquigarrow \sigma'}{\langle c_0 \,;\, c_1, \sigma\rangle \rightsquigarrow \langle c_1, \sigma'\rangle}$$

$$\frac{\langle c_0, \sigma\rangle \rightsquigarrow \langle c_0', \sigma'\rangle}{\langle c_0 \,;\, c_1, \sigma\rangle \rightsquigarrow \langle c_0' \,;\, c_1, \sigma'\rangle}$$

For example:

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

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

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

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

Conditional and **while** Commands

- Conditional

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

$$\frac{}{\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma\rangle \rightsquigarrow \langle c_0, \sigma\rangle}$$

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

$$\frac{}{\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma\rangle \rightsquigarrow \langle c_1, \sigma\rangle}$$

- **while**

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

$$\frac{}{\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma\rangle \rightsquigarrow \langle c\ ;\ \mathbf{while}\ b\ \mathbf{do}\ c, \sigma\rangle}$$

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

$$\frac{}{\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma\rangle \rightsquigarrow \sigma}$$

Exercise 2.1

Let c be the command

$$\textbf{while } \mathsf{x} \neq 0 \textbf{ do if } \mathsf{x} = 1 \textbf{ then skip else } \mathsf{x} := \mathsf{x} - 2.$$

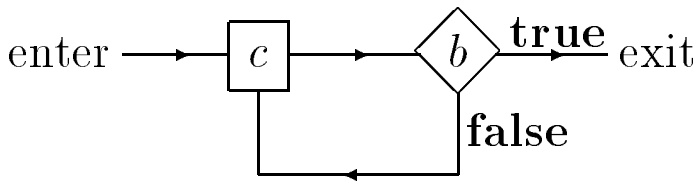
1. Give an execution that begins with the configuration

$$\langle c, [\,\mathsf{x}\colon 2\,]\rangle.$$

2. Prove the second transition of this execution.

Exercise 2.2

A **repeat** command has the form **repeat** c **until** b. Its effect is described by the following flowchart:



Describe the syntax and semantics of this command by giving an abstract-grammar production and transition rules.

Specifications:

Syntax

$\langle \text{spec} \rangle ::= \{\langle \text{precedent} \rangle\}\ \langle \text{comm} \rangle\ \{\langle \text{consequent} \rangle\}$ (partial correctness)

$\quad | \ [\ \langle \text{precedent} \rangle\ ]\ \langle \text{comm} \rangle\ [\ \langle \text{consequent} \rangle\ ]$ (total correctness)

$\langle \text{precedent} \rangle ::= \langle \text{assert} \rangle$

$\langle \text{consequent} \rangle ::= \langle \text{assert} \rangle$

Semantics

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

$[\![\{p\}\ c\ \{q\}]\!]_{\text{spec}}$ iff $\forall \sigma \in \text{States}.\ [\![p]\!]_{\text{assert}}\sigma$ implies
$(\forall \sigma'.\ c, \sigma \rightsquigarrow^* \sigma'$ implies $[\![q]\!]_{\text{assert}}\sigma')$

$[\![[\ p\ ]\ c\ [\ q\ ]]\!]_{\text{spec}}$ iff $\forall \sigma \in \text{States}.\ [\![p]\!]_{\text{assert}}\sigma$ implies
$\neg\ c, \sigma \uparrow$
and $(\forall \sigma'.\ c, \sigma \rightsquigarrow^* \sigma'$ implies $[\![q]\!]_{\text{assert}}\sigma')$.

Examples of Partial Correctness Specifications

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

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

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

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

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

Examples of Total Correctness Specifications

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

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

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

Inference Rules:

Assignment (AS)

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

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:

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

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

Inference Rules:

Strengthening Precedent (SP)

$$\frac{p \Rightarrow q \qquad \{q\}\ c\ \{r\}}{\{p\}\ c\ \{r\}} \qquad \frac{p \Rightarrow q \qquad [\,q\,]\ c\ [\,r\,]}{[\,p\,]\ c\ [\,r\,]}$$

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)

$$\frac{\{p\}\ c\ \{q\} \qquad q \Rightarrow r}{\{p\}\ c\ \{r\}} \qquad \frac{[\,p\,]\ c\ [\,q\,] \qquad q \Rightarrow r}{[\,p\,]\ c\ [\,r\,]}$$

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:

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:

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:

skip (SK)

$$\frac{}{\{p\}\ \mathbf{skip}\ \{p\}} \qquad \frac{}{[\,p\,]\ \mathbf{skip}\ [\,p\,]}$$

Conditional (CD)

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

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

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)

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

Conditional (CD)

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

Rules for Annotated Specifications:

Partial Correctness of **while** (WHP)

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

Total Correctness of **while** (WHT)

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

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

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)

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

Proof of an Annotated Specification

1. $(\mathsf{n} \geq 0 \wedge \mathsf{n} = 0) \Rightarrow 0 = \mathrm{fib}(\mathsf{n})$ (VC)

2. $[\, \mathsf{n} \geq 0 \wedge \mathsf{n} = 0 \,]\ \mathsf{f} := 0\ [\, \mathsf{f} = \mathrm{fib}(\mathsf{n}) \,]$ (AS1,1)

3. $(\mathsf{n} \geq 0 \wedge \neg\ \mathsf{n} = 0)$
   $\Rightarrow (1 = \mathrm{fib}(1) \wedge 0 = \mathrm{fib}(1-1) \wedge 1 \leq \mathsf{n})$ (VC)

4. $[\mathsf{n} \geq 0 \wedge \neg\ \mathsf{n} = 0]$
   $\mathsf{k} := 1$
   $[1 = \mathrm{fib}(\mathsf{k}) \wedge 0 = \mathrm{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n}]$ (AS1,3)

5. $[\mathsf{n} \geq 0 \wedge \neg\ \mathsf{n} = 0]$
   $\mathsf{k} := 1\ ;\ \mathsf{g} := 0$
   $[1 = \mathrm{fib}(\mathsf{k}) \wedge \mathsf{g} = \mathrm{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n}]$ (AS2,4)

6. $[\mathsf{n} \geq 0 \wedge \neg\ \mathsf{n} = 0]$
   $\mathsf{k} := 1\ ;\ \mathsf{g} := 0\ ;\ \mathsf{f} := 1$
   $[\mathsf{f} = \mathrm{fib}(\mathsf{k}) \wedge \mathsf{g} = \mathrm{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n}]$ (AS2,5)

Proof of an Annotated Specification (continued)

7. $(\mathsf{f} = \mathrm{fib}(\mathsf{k}) \wedge \mathsf{g} = \mathrm{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n} \wedge \mathsf{k} \neq \mathsf{n} \wedge \mathsf{n} - \mathsf{k} = \mathsf{v}_0)$
   $\Rightarrow (\mathsf{f} + \mathsf{g} = \mathrm{fib}(\mathsf{k}+1) \wedge \mathsf{f} = \mathrm{fib}((\mathsf{k}+1)-1) \wedge \mathsf{k}+1 \leq \mathsf{n}$
   $\wedge\, \mathsf{n} - (\mathsf{k}+1) < \mathsf{v}_0)$ (VC)

8. $[\mathsf{f} = \mathrm{fib}(\mathsf{k}) \wedge \mathsf{g} = \mathrm{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n} \wedge \mathsf{k} \neq \mathsf{n} \wedge \mathsf{n} - \mathsf{k} = \mathsf{v}_0]$
   $\mathsf{t} := \mathsf{g}$
   $[\mathsf{f} + \mathsf{t} = \mathrm{fib}(\mathsf{k}+1) \wedge \mathsf{f} = \mathrm{fib}((\mathsf{k}+1)-1) \wedge \mathsf{k}+1 \leq \mathsf{n}$
   $\wedge\, \mathsf{n} - (\mathsf{k}+1) < \mathsf{v}_0]$ (AS1,7)

9. $[\mathsf{f} = \mathrm{fib}(\mathsf{k}) \wedge \mathsf{g} = \mathrm{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n} \wedge \mathsf{k} \neq \mathsf{n} \wedge \mathsf{n} - \mathsf{k} = \mathsf{v}_0]$
   $\mathsf{t} := \mathsf{g}\,;\, \mathsf{g} := \mathsf{f}$
   $[\mathsf{f} + \mathsf{t} = \mathrm{fib}(\mathsf{k}+1) \wedge \mathsf{g} = \mathrm{fib}((\mathsf{k}+1)-1) \wedge \mathsf{k}+1 \leq \mathsf{n}$
   $\wedge\, \mathsf{n} - (\mathsf{k}+1) < \mathsf{v}_0]$ (AS2,8)

10. $[\mathsf{f} = \mathrm{fib}(\mathsf{k}) \wedge \mathsf{g} = \mathrm{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n} \wedge \mathsf{k} \neq \mathsf{n} \wedge \mathsf{n} - \mathsf{k} = \mathsf{v}_0]$
   $\mathsf{t} := \mathsf{g}\,;\, \mathsf{g} := \mathsf{f}\,;\, \mathsf{f} := \mathsf{f} + \mathsf{t}$
   $[\mathsf{f} = \mathrm{fib}(\mathsf{k}+1) \wedge \mathsf{g} = \mathrm{fib}((\mathsf{k}+1)-1) \wedge \mathsf{k}+1 \leq \mathsf{n}$
   $\wedge\, \mathsf{n} - (\mathsf{k}+1) < \mathsf{v}_0]$ (AS2,9)

11. $[\mathsf{f} = \mathrm{fib}(\mathsf{k}) \wedge \mathsf{g} = \mathrm{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n} \wedge \mathsf{k} \neq \mathsf{n} \wedge \mathsf{n} - \mathsf{k} = \mathsf{v}_0]$
   $\mathsf{t} := \mathsf{g}\,;\, \mathsf{g} := \mathsf{f}\,;\, \mathsf{f} := \mathsf{f} + \mathsf{t}\,;\, \mathsf{k} := \mathsf{k}+1$
   $[\mathsf{f} = \mathrm{fib}(\mathsf{k}) \wedge \mathsf{g} = \mathrm{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n} \wedge \mathsf{n} - \mathsf{k} < \mathsf{v}_0]$
   (AS2,10)

Proof of an Annotated Specification (continued)

12. $(\mathsf{f} = \mathrm{fib}(\mathsf{k}) \wedge \mathsf{g} = \mathrm{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n} \wedge \mathsf{k} \neq \mathsf{n}) \Rightarrow \mathsf{n} - \mathsf{k} \geq 0$
(VC)

13. $(\mathsf{f} = \mathrm{fib}(\mathsf{k}) \wedge \mathsf{g} = \mathrm{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n} \wedge \neg\ \mathsf{k} \neq \mathsf{n}) \Rightarrow \mathsf{f} = \mathrm{fib}(\mathsf{n})$
(VC)

14. $[\mathsf{f} = \mathrm{fib}(\mathsf{k}) \wedge \mathsf{g} = \mathrm{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n}]$
while $[\mathbf{vrnt}\colon \mathsf{n} - \mathsf{k}]\, \mathsf{k} \neq \mathsf{n}$ **do**
$(\mathsf{t} := \mathsf{g}\ ;\ \mathsf{g} := \mathsf{f}\ ;\ \mathsf{f} := \mathsf{f} + \mathsf{t}\ ;\ \mathsf{k} := \mathsf{k} + 1)$
$[\mathsf{f} = \mathrm{fib}(\mathsf{n})]$ (WHT,11,12,13)

15. $[\mathsf{n} \geq 0 \wedge \neg\ \mathsf{n} = 0]$
$\mathsf{k} := 1\ ;\ \mathsf{g} := 0\ ;\ \mathsf{f} := 1\ ;$
$[\mathsf{f} = \mathrm{fib}(\mathsf{k}) \wedge \mathsf{g} = \mathrm{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n}]$
while $[\mathbf{vrnt}\colon \mathsf{n} - \mathsf{k}]\, \mathsf{k} \neq \mathsf{n}$ **do**
$(\mathsf{t} := \mathsf{g}\ ;\ \mathsf{g} := \mathsf{f}\ ;\ \mathsf{f} := \mathsf{f} + \mathsf{t}\ ;\ \mathsf{k} := \mathsf{k} + 1)$
$[\mathsf{f} = \mathrm{fib}(\mathsf{n})]$ (SQ,6,14)

16. $[\mathsf{n} \geq 0]$
if $\mathsf{n} = 0$ **then** $\mathsf{f} := 0$ **else**
$(\mathsf{k} := 1\ ;\ \mathsf{g} := 0\ ;\ \mathsf{f} := 1\ ;$
$[\mathsf{f} = \mathrm{fib}(\mathsf{k}) \wedge \mathsf{g} = \mathrm{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n}]$
while $[\mathbf{vrnt}\colon \mathsf{n} - \mathsf{k}]\, \mathsf{k} \neq \mathsf{n}$ **do**
$(\mathsf{t} := \mathsf{g}\ ;\ \mathsf{g} := \mathsf{f}\ ;\ \mathsf{f} := \mathsf{f} + \mathsf{t}\ ;\ \mathsf{k} := \mathsf{k} + 1))$
$[\mathsf{f} = \mathrm{fib}(\mathsf{n})]$ (CD,2,15)

An Annotated Program for Fast Exponentiation

$$
\begin{array}{l}
[\mathsf{n} \geq 0] \\
\mathsf{k} := \mathsf{n} \,;\, \mathsf{z} := \mathsf{x} \,;\, \mathsf{y} := 1; \\
[\mathsf{y} \times \mathsf{z}^{\mathsf{k}} = \mathsf{x}^{\mathsf{n}} \wedge \mathsf{k} \geq 0] \\
\textbf{while } [\textbf{vrnt}\colon \mathsf{k}]\, \mathsf{k} \neq 0 \textbf{ do} \\
\quad \textbf{if even } \mathsf{k} \textbf{ then } (\mathsf{k} := \mathsf{k} \div 2 \,;\, \mathsf{z} := \mathsf{z} \times \mathsf{z}) \\
\qquad \textbf{else } (\mathsf{k} := \mathsf{k} - 1 \,;\, \mathsf{y} := \mathsf{y} \times \mathsf{z}) \\
[\mathsf{y} = \mathsf{x}^{\mathsf{n}}]
\end{array}
$$

Verification Conditions

$$(\mathsf{n} \geq 0) \Rightarrow (1 \times \mathsf{x}^{\mathsf{n}} = \mathsf{x}^{\mathsf{n}} \wedge \mathsf{n} \geq 0)$$

$$\Big(\mathsf{y} \times \mathsf{z}^{\mathsf{k}} = \mathsf{x}^{\mathsf{n}} \wedge \mathsf{k} \geq 0 \wedge \mathsf{k} \neq 0 \wedge \mathsf{k} = \mathsf{k}_0 \wedge \textbf{even } \mathsf{k}\Big) \Rightarrow \Big(\mathsf{y} \times (\mathsf{z} \times \mathsf{z})^{\mathsf{k} \div 2} = \mathsf{x}^{\mathsf{n}} \wedge \mathsf{k} \div 2 \geq 0 \wedge \mathsf{k} \div 2 < \mathsf{k}_0\Big)$$

$$\Big(\mathsf{y} \times \mathsf{z}^{\mathsf{k}} = \mathsf{x}^{\mathsf{n}} \wedge \mathsf{k} \geq 0 \wedge \mathsf{k} \neq 0 \wedge \mathsf{k} = \mathsf{k}_0 \wedge \neg\, \textbf{even } \mathsf{k}\Big) \Rightarrow \Big((\mathsf{y} \times \mathsf{z}) \times \mathsf{z}^{\mathsf{k} - 1} = \mathsf{x}^{\mathsf{n}} \wedge \mathsf{k} - 1 \geq 0 \wedge \mathsf{k} - 1 < \mathsf{k}_0\Big)$$

$$(\mathsf{y} \times \mathsf{z}^{\mathsf{k}} = \mathsf{x}^{\mathsf{n}} \wedge \mathsf{k} \geq 0 \wedge \mathsf{k} \neq 0) \Rightarrow \mathsf{k} \geq 0$$

$$(\mathsf{y} \times \mathsf{z}^{\mathsf{k}} = \mathsf{x}^{\mathsf{n}} \wedge \mathsf{k} \geq 0 \wedge \neg\, \mathsf{k} \neq 0) \Rightarrow \mathsf{y} = \mathsf{x}^{\mathsf{n}}$$

Additional Inference Rules:

Variable Declaration (DC)

$$\frac{\{p\}\ c_1\ ;\cdots;\ c_n\ ;\ v := e\ ;\ c\ \{q\}}{\{p\}\ c_1\ ;\cdots;\ c_n\ ;\ \textbf{newvar}\ v := e\ \textbf{in}\ c\ \{q\}}$$

$$\frac{[\,p\,]\ c_1\ ;\cdots;\ c_n\ ;\ v := e\ ;\ c\ [\,q\,]}{[\,p\,]\ c_1\ ;\cdots;\ c_n\ ;\ \textbf{newvar}\ v := e\ \textbf{in}\ c\ [\,q\,]}$$

when v does not occur free in q and $n \geq 0$.

Here the requirement on v formalizes the concept of *locality*, i.e., that the value of the declared variable plays no further role after execution of the scope of the declaration. 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** $\mathsf{t} := \mathsf{x} + \mathsf{y}$ **in** $\mathsf{z} := \mathsf{t} \times \mathsf{t}$ meets only the first specification:

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

Additional Inference Rules:

Renaming (RN)

$$\frac{\{p\}\ c\ \{q\}}{\{p'\}\ c'\ \{q'\}} \qquad \frac{[\,p\,]\ c\ [\,q\,]}{[\,p'\,]\ c'\ [\,q'\,]},$$

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

Changing Free Variables (CF)

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

where $v_1, \ldots, v_n$ are the variables occurring free in p, c, or q, and $v_1', \ldots, v_n'$ are distinct.

Additional Inference Rules:

Conjunction of Assertions (CA)

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

Disjunction of Assertions (DA)

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

Constancy for Partial Correctness (CSP)

$$\frac{}{\{p\}\ c\ \{p\}}$$

when no variable occurs both free in p and free on the left of an assignment command in c.

Constancy for Total Correctness (CST)

$$\frac{[\,p\,]\ c\ [\,q\,]}{[\,p \wedge r\,]\ c\ [\,q \wedge r\,]}$$

when no variable occurs both free in r and free on the left of an assignment command in c.

Exercise 3.1

Fill in the missing precedents indicated by question marks to make the following specifications true. The assertions that you provide should be as weak as possible, but they should not be unnecessarily complicated.

$[\,?\,]\ \mathsf{x} := \mathsf{x} + 1\ ;\ \mathsf{y} := \mathsf{y} + 2 \times \mathsf{x} - 1\ [\,\mathsf{y} = \mathsf{x}^2\,]$

$[\,?\,]\ \mathsf{x} := \mathsf{x} + \mathsf{y}\ ;\ \mathsf{y} := \mathsf{x} - \mathsf{y}\ ;\ \mathsf{x} := \mathsf{x} - \mathsf{y}\ [\,\mathsf{x} = \mathsf{z} \wedge \mathsf{y} = \mathsf{w}\,]$

$[\,?\,]$ **while** $\mathsf{a} < \mathsf{b}$ **do** $(\mathsf{a} := \mathsf{a} + 1\ ;\ \mathsf{y} := \mathsf{x} + \mathsf{y})\ [\,\mathsf{y} = \mathsf{x} \times \mathsf{b}\,]$

$\{?\}$ **while true do skip** $\{$**false**$\}$

$[\,?\,]$ **while true do skip** $[\,$**false**$\,]$.

Exercise 3.2: Logarithmic-Time Multiplication

Complete the following "skeleton" by replacing the remarks in quotation marks. The result should be a correct annotated total-correctness specification (of a program that does not assign to the input variables x or y). You may use multiplication or division by two as a primitive operation but not multiplication or division by an arbitrary number.

$[\mathsf{y} \geq 0]$

"Achieve invariant" ;

$[\mathsf{z} + \mathsf{w} \times \mathsf{k} = \mathsf{x} \times \mathsf{y} \wedge \mathsf{k} \geq 0]$

while $[\mathbf{vrnt}\colon \mathsf{k}]\, \mathsf{k} \neq 0$ **do** "Preserve invariant"

$[\mathsf{z} = \mathsf{x} \times \mathsf{y}]$

Give the verification conditions needed to prove the annotated specification.

Exercise 3.3: Logarithmic-Time Division

Complete the following "skeleton" by replacing the remarks in quotation marks. The result should be a correct annotated total-correctness specification (of a program that does not assign to the input variables x or y). You may use multiplication or division by two as a primitive operation but not multiplication or division by an arbitrary number.

$[\mathsf{x} \geq 0 \wedge \mathsf{y} > 0]$

"Achieve first invariant" ;

$[\mathsf{z} = \mathsf{y} \times 2^{\mathsf{n}} \wedge \mathsf{n} \geq 0 \wedge \mathsf{x} \geq 0 \wedge \mathsf{y} > 0]$

while $[\mathbf{vrnt}\colon \mathsf{x} - \mathsf{z}]\, \mathsf{z} \leq \mathsf{x}$ **do** "Preserve first invariant" ;

"Achieve second invariant" ;

$[\mathsf{x} = \mathsf{q} \times \mathsf{z} + \mathsf{r} \wedge 0 \leq \mathsf{r} < \mathsf{z} \wedge \mathsf{z} = \mathsf{y} \times 2^{\mathsf{n}} \wedge \mathsf{n} \geq 0]$

while $[\mathbf{vrnt}\colon \mathsf{n}]\, \mathsf{n} \neq 0$ **do** "Preserve second invariant"

$[\mathsf{x} = \mathsf{q} \times \mathsf{y} + \mathsf{r} \wedge 0 \leq \mathsf{r} < \mathsf{y}]$

Give the verification conditions needed to prove the annotated specification.

An Extension of the Simple Imperative Language

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

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

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

Why cons and . do not occur in expressions

$(\mathsf{x} = \mathsf{x})/\mathsf{x} \rightarrow \mathbf{cons}(1,2)$ is

$$\mathbf{cons}(1,2) = \mathbf{cons}(1,2)$$

$(\mathsf{x} = \mathsf{x} * \mathsf{y} = \mathsf{y})/\mathsf{x} \rightarrow [\mathsf{z}], \mathsf{y} \rightarrow [\mathsf{z}]$ is

$$([\mathsf{z}] = [\mathsf{z}]) * ([\mathsf{z}] = [\mathsf{z}]).$$

Stores and Heaps

$$\begin{aligned}
\text{Atoms} &\subseteq \mathbf{Z} \\
\text{Locations} &\subseteq \mathbf{Z} \\
\text{Atoms} \cap \text{Locations} &= \{\} \\
\mathbf{nil} &\in \text{Atoms} \\
\\
\text{States} &= \text{Stores} \times \text{Heaps} \\
\text{Stores} &= (\langle \text{var} \rangle \to \mathbf{Z}) \\
\text{Heaps} &= \bigcup_{A \subseteq \text{Locations}}^{\text{fin}} (A \to \mathbf{Z}) \\
\\
[\![-]\!]_{\text{exp}} &\in \langle \text{exp} \rangle \to \text{Stores} \to \mathbf{B} \\
[\![-]\!]_{\text{boolexp}} &\in \langle \text{boolexp} \rangle \to \text{Stores} \to \mathbf{B} \\
\Gamma_N &= \langle \text{comm} \rangle \times \text{States} \\
\Gamma_T &= \text{States} \cup \{\mathbf{abort}\}
\end{aligned}$$

Transition Rules

- 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

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

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

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

Conditional and **while** 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 $[\![b]\!]_{\text{boolexp}} s = \mathbf{false}$:

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

New Commands

- Construction

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

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

- Lookup

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

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

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

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

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

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

Pure Commands

A command c is *pure* if its execution is independent of the heap, i.e.,

$$\forall s, s' \in \text{Stores}.\ \forall h, h' \in \text{Heaps}.$$
$$\langle c, (s, h)\rangle \rightsquigarrow^* (s', h') \text{ implies } h' = h$$

and

$$\forall s, s' \in \text{Stores}.\ \forall h_1, h_2 \in \text{Heaps}.$$
$$(\langle c, (s, h_1)\rangle \rightsquigarrow^* \mathbf{abort} \text{ iff } \langle c, (s, h_2)\rangle \rightsquigarrow^* \mathbf{abort})$$
$$\text{and } (\langle c, (s, h_1)\rangle \uparrow \text{ iff } \langle c, (s, h_2)\rangle \uparrow)$$
$$\text{and } (\langle c, (s, h_1)\rangle \rightsquigarrow^* (s', h_1) \text{ iff } \langle c, (s, h_2)\rangle \rightsquigarrow^* (s', h_2)).$$

A command is *syntactically pure* if it does not contain any of the new commands

$$v := \mathbf{cons}(e_0, \ldots, e_{n-1}) \qquad v := [e] \qquad [e] := e' \qquad \mathbf{dispose}\ e.$$

A syntactically pure command is pure.

An Extension of Assertions

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

The Semantics of Assertions

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

$$\begin{aligned}
[\![\mathbf{true}]\!]_{\text{assert}}(s,h) &= \mathbf{true}, \\
[\![\mathbf{false}]\!]_{\text{assert}}(s,h) &= \mathbf{false}, \\
[\![e_0 = e_1]\!]_{\text{assert}}(s,h) &= ([\![e_0]\!]_{\text{exp}} s = [\![e_1]\!]_{\text{exp}} s) \\
&\qquad \text{(and similarly for } \neq, \ldots), \\
[\![\neg\, p]\!]_{\text{assert}}(s,h) &= \neg [\![p]\!]_{\text{assert}}(s,h), \\
[\![p_0 \wedge p_1]\!]_{\text{assert}}(s,h) &= [\![p_0]\!]_{\text{assert}}(s,h) \wedge [\![p_1]\!]_{\text{assert}}(s,h) \\
&\qquad \text{(and similarly for } \vee, \Rightarrow, \Leftrightarrow), \\
[\![\forall v.\ p]\!]_{\text{assert}}(s,h) &= \forall x \in \mathbf{Z}.\ [\![p]\!]_{\text{assert}}([\, s \mid v{:}\, x \,], h), \\
[\![\exists v.\ p]\!]_{\text{assert}}(s,h) &= \exists x \in \mathbf{Z}.\ [\![p]\!]_{\text{assert}}([\, s \mid v{:}\, x \,], h).
\end{aligned}$$

New Assertions

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

$$[\![\mathbf{emptyheap}]\!]_{\text{assert}}(s,h)$$
$$\text{iff } \operatorname{dom} h = \{\},$$

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

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

where

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

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

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

Pure Assertions

An assertion p is *pure* if its meaning is not affected by the heap, i.e., if

$$\forall s \in \text{Stores}.\ \exists b \in \mathbf{B}.\ \forall h \in \text{Heaps}.\ [\![p]\!]_{\text{assert}}(s,h) = b.$$

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

A syntactically pure assertion is pure.

Some Abbreviations

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

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

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

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

$$\text{iff } e \mapsto e_0, \ldots, e_{n-1} * \mathbf{true}$$

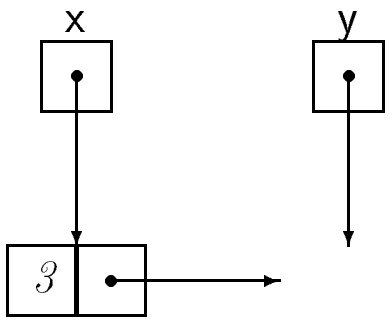
An Example of $*$

$$[\![p_0 * p_1]\!]_{\text{assert}}(s,h)$$

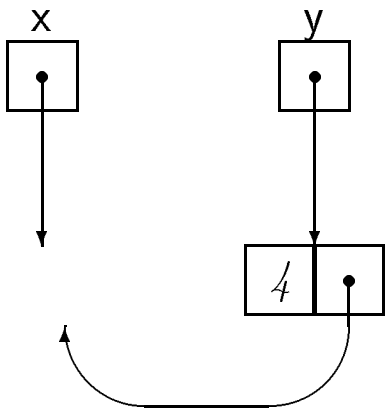
$$\text{iff } \exists h_0, h_1.\ h_0 \perp h_1 \text{ and } h_0 \cdot h_1 = h$$

$$\text{and } [\![p_0]\!]_{\text{assert}}(s,h_0) \text{ and } [\![p_1]\!]_{\text{assert}}(s,h_1)$$

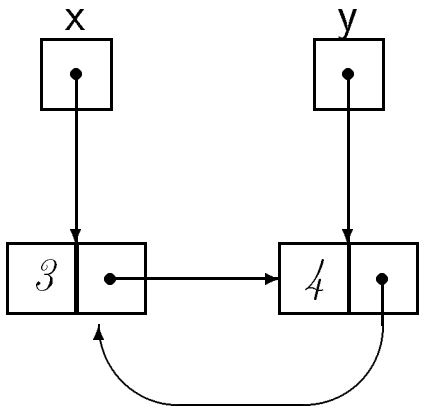
$\mathsf{x} \mapsto 3, \mathsf{y}$



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



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



More Examples of $*$

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

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

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**
$\mathsf{x} \mapsto 1 \ * \ \mathbf{true}$	$h_1 \subseteq h$
$\mathsf{x} \mapsto 1 \ * \ \neg\, \mathsf{x} \mapsto 1$	$h_1 \subseteq h$

Axiom Schemata for $*$

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

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

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

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

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

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

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

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

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

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

An Inference Rule

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

Axiom Schemata for $\mapsto$

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

$$(x \mapsto e * \mathbf{true}) \wedge \mathbf{emptyheap} \Leftrightarrow \mathbf{false}$$

$$x \mapsto e * x' \mapsto e' \Rightarrow x \neq x'.$$

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

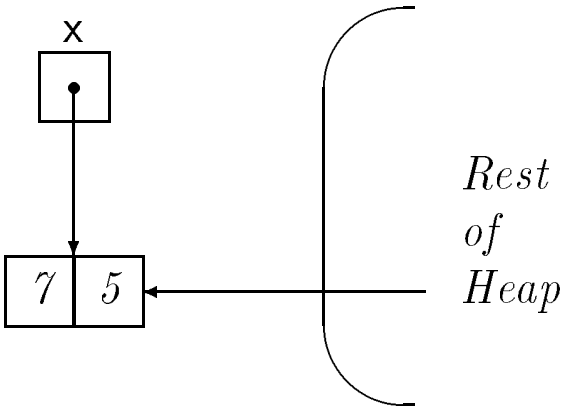
$q : \mathsf{y} \mapsto 2$

An Example of $-\!\!*$

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

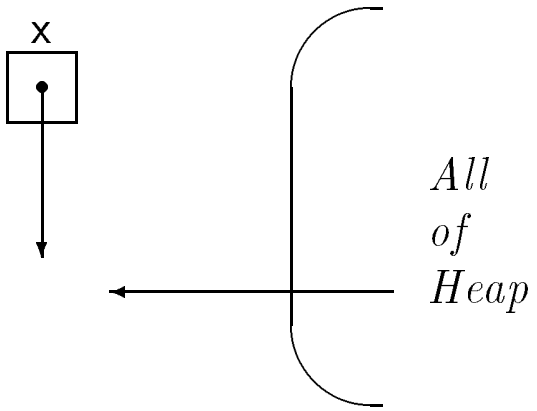
If:

P



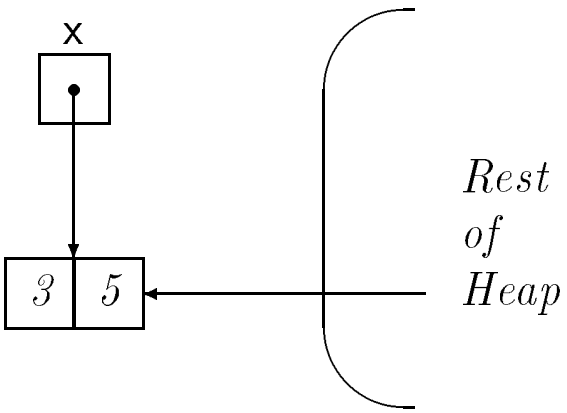
then:

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



and:

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



Axiom Schemata for $-\!\!*$

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

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

Inference Rules

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

Derived Inference Rules

$$\frac{p_1 \Rightarrow (q \multimap r) \qquad p_2 \Rightarrow (r \multimap s)}{p_2 \ * \ p_1 \Rightarrow (q \multimap s)}$$

$$\dfrac{\dfrac{\dfrac{}{p_2 \Rightarrow p_2} \qquad \dfrac{p_1 \Rightarrow (q \multimap r)}{p_1 \ * \ q \Rightarrow r}}{p_2 \ * \ p_1 \ * \ q \Rightarrow p_2 \ * \ r} \qquad \dfrac{p_2 \Rightarrow (r \multimap s)}{p_2 \ * \ r \Rightarrow s}}{\dfrac{p_2 \ * \ p_1 \ * \ q \Rightarrow s}{p_2 \ * \ p_1 \Rightarrow (q \multimap s)}}$$

$$\frac{p' \Rightarrow p \qquad q \Rightarrow q'}{(p \multimap q) \Rightarrow (p' \multimap q').}$$

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

Specifications

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

$[\![\{p\}\ c\ \{q\}]\!]_{\text{spec}}$ iff $\forall \sigma \in \text{States}.\ [\![p]\!]_{\text{assert}}\sigma$ implies

$\neg\ c, \sigma \rightsquigarrow^* \mathbf{abort}$

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

$[\![[\ p\ ]\ c\ [\ q\ ]]\!]_{\text{spec}}$ iff $\forall \sigma \in \text{States}.\ [\![p]\!]_{\text{assert}}\sigma$ implies

$\neg\ c, \sigma \rightsquigarrow^* \mathbf{abort}$

and $\neg\ c, \sigma \uparrow$

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

Inference Rules for Mutation

1.
$$\{p \ * \ (e \mapsto -)\}$$
$$[e] := e'$$
$$\{p \ * \ (e \mapsto e')\}$$

2.
$$\{((e \mapsto e') \mathrel{-\!\!*} p) \ * \ (e \mapsto -)\}$$
$$[e] := e'$$
$$\{p\}$$

Inference Rules for Construction

For any phrase p, we write p' for the substitution $p/v \to v'$.

1.
$$\{p\}$$
$$v := \mathbf{cons}(e_0, \ldots, e_{n-1})$$
$$\{p \ * \ (v \mapsto e_0, \ldots, e_{n-1})\}$$

where v is not free in $e_0, \ldots, e_{n-1}$, or p.

2.
$$\{p\}$$
$$v := \mathbf{cons}(e_0, \ldots, e_{n-1})$$
$$\{\exists v'.\ p' \ * \ (v \mapsto e'_0, \ldots, e'_{n-1})\}$$

where v' is distinct from v and not free in $e_0, \ldots, e_{n-1}$, or p.

3.
$$\{\forall v'.\ (v' \mapsto e_0, \ldots, e_{n-1}) \mathrel{-\!\!*} p'\}$$
$$v := \mathbf{cons}(e_0, \ldots, e_{n-1})$$
$$\{p\}$$

where v' is not free in $e_0, \ldots, e_{n-1}$, or p.

An Inference Rule for Lookup

$$\begin{array}{l}\{\exists v'.\ p' \wedge e \hookrightarrow v'\} \\ v := [e] \\ \{p\}\end{array}$$

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

Disposal

$$\begin{array}{l}\{p\ *\ (e \mapsto -)\} \\ \textbf{dispose}\ e \\ \{p\}\end{array}$$

Frame Introduction (Constancy Revisited)

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

when no variable occurs both free in r and free on the left of an assignment command in c.

For all of the new inference rules, there are similar rules for total correctness.

Exercise 4.1

None of the following axiom schemata are sound. For each, given an instance which is not valid.

$$p_1 \wedge p_2 \not\Rightarrow p_1 \ * \ p_2$$

$$p_1 \ * \ p_2 \not\Rightarrow p_1 \wedge p_2$$

$$(p_1 \ * \ q) \wedge (p_2 \ * \ q) \not\Rightarrow (p_1 \wedge p_2) \ * \ q$$

$$(p_1 \vee q) \ * \ (p_2 \vee q) \not\Rightarrow (p_1 \ * \ p_2) \vee q$$

$$(p_1 \ * \ p_2) \vee q \not\Rightarrow (p_1 \vee q) \ * \ (p_2 \vee q)$$

$$(p_1 \wedge q) \ * \ (p_2 \wedge q) \not\Rightarrow (p_1 \ * \ p_2) \wedge q$$

$$(p_1 \ * \ p_2) \wedge q \not\Rightarrow (p_1 \wedge q) \ * \ (p_2 \wedge q)$$

$$(p_1 \Rightarrow p_2) \not\Rightarrow (p_1 \ * \ q) \Rightarrow (p_2 \ * \ q)$$

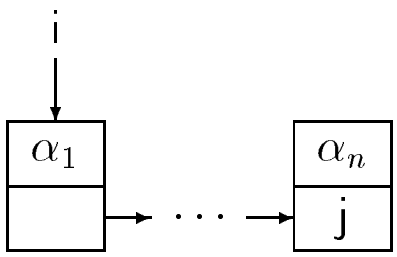
$$\forall x.\, (p_1 \ * \ p_2) \not\Rightarrow (\forall x.\, p_1) \ * \ p_2 \quad \text{when } x \text{ not free in } p_2$$

$$(p_1 \mathrel{-\!\!*} p_2) \not\Rightarrow (p_1 \Rightarrow p_2)$$

$$(p_1 \Rightarrow p_2) \not\Rightarrow (p_1 \mathrel{-\!\!*} p_2)$$

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:

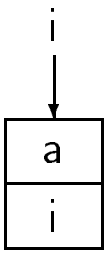
$$\begin{aligned}
&\exists \mathsf{j}.\ \mathsf{list}\ \epsilon\ (\mathsf{i},\mathsf{j})\ *\ \mathsf{list}\ \beta\ (\mathsf{j},\mathsf{k}) \\
&\quad \Leftrightarrow \exists \mathsf{j}.\ (\mathbf{emptyheap} \wedge \mathsf{i} = \mathsf{j})\ *\ \mathsf{list}\ \beta\ (\mathsf{j},\mathsf{k}) \\
&\quad \Leftrightarrow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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Exercise 5.1

When

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

we say that j is an *interior pointer* of the list segment described by $\mathbf{list}\ \gamma\ (\mathsf{i}, \mathsf{k})$.

1. Give an assertion describing a list segment with two interior pointers j_1 and j_2, such that j_1 comes before than, or at the same point as, j_2 in the ordering of the elements of the list segment.

2. Give an assertion describing a list segment with two interior pointers j_1 and j_2, where there is no constraint on the relative positions of j_1 and j_2.

3. Prove that the first assertion implies the second.

Exercise 5.2

A *braced list segment* is a list segment with an interior pointer j to its last element; in the special case where the list segment is empty, j is **nil**. Formally,

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

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

1. Prove the assertion

$$\mathbf{brlist}\ \alpha\,(\mathsf{i},\mathsf{j},\mathsf{k}) \Rightarrow \mathbf{list}\ \alpha\,(\mathsf{i},\mathsf{k}).$$

2. Give an annotated specification of a command that changes the final pointer of a braced list segment from k to l. The command should not construct or deallocate any list structure, and it should not change the value of j. It should satisfy

$$\{\mathbf{brlist}\ \alpha\,(\mathsf{i},\mathsf{j},\mathsf{k})\}\ \cdots\ \{\mathbf{brlist}\ \alpha\,(\mathsf{i},\mathsf{j},\mathsf{l})\}.$$

3. Give an annotated specification of a command that concatenates two braced list segments. The command should not construct or deallocate any list structure, and it should not change the value of k_2. It should satisfy

$$\{\mathbf{brlist}\ \alpha\,(\mathsf{i}_1,\mathsf{j}_1,\mathsf{k}_1)\ *\ \mathbf{brlist}\ \beta\,(\mathsf{i}_2,\mathsf{j}_2,\mathsf{k}_2)\}$$
$$\ldots$$
$$\{\mathbf{brlist}\ \alpha{\cdot}\beta\,(\mathsf{i}_1,\mathsf{j},\mathsf{k}_2)\}.$$

Exercise 5.2 (continued)

4. Give an annotated specification of a command that appends an item a at the beginning of a braced list segment. The command should not change the value of k, and should satisfy

$$\{\mathbf{brlist}\ \alpha\,(\mathsf{i},\mathsf{j},\mathsf{k})\}\ \cdots\ \{\mathbf{brlist}\ \mathsf{a}{\cdot}\alpha\,(\mathsf{i},\mathsf{j},\mathsf{k})\}.$$

5. Give an annotated specification of a command that appends an item a at the end of a braced list segment. The command should not change the value of k, and should satisfy

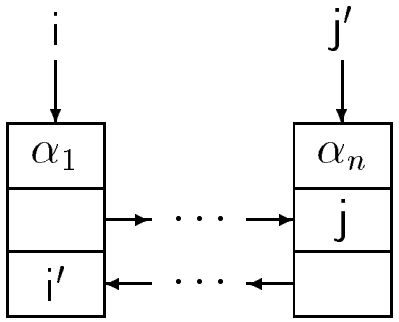
$$\{\mathbf{brlist}\ \alpha\,(\mathsf{i},\mathsf{j},\mathsf{k})\}\ \cdots\ \{\mathbf{brlist}\ \alpha{\cdot}\mathsf{a}\,(\mathsf{i},\mathsf{j},\mathsf{k})\}.$$

6. Give an annotated specification of a command that deletes an item a from the beginning of a nonempty braced list segment. The command should not change the value of k, and should satisfy

$$\{\mathbf{brlist}\ \mathsf{a}{\cdot}\alpha\,(\mathsf{i},\mathsf{j},\mathsf{k})\}\ \cdots\ \{\mathbf{brlist}\ \alpha\,(\mathsf{i},\mathsf{j},\mathsf{k})\}.$$

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

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

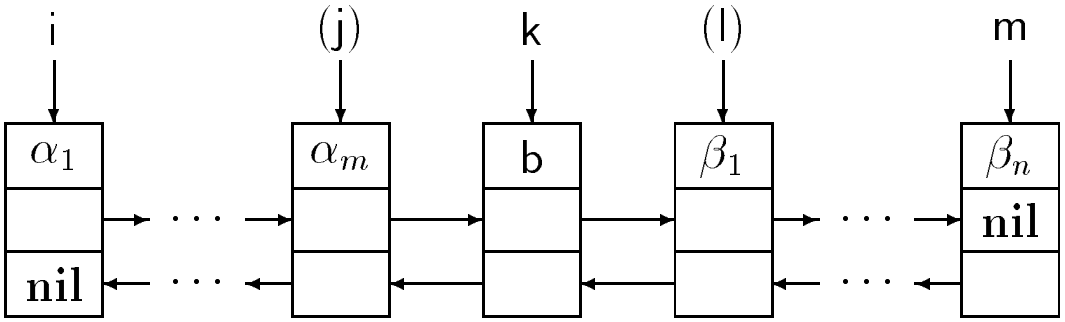
$$\mathsf{dlist}\,\alpha{\cdot}\beta\,(\mathsf{i},\mathsf{i}',\mathsf{k},\mathsf{k}') \Leftrightarrow \exists \mathsf{j},\mathsf{j}'.\ \mathsf{dlist}\,\alpha\,(\mathsf{i},\mathsf{i}',\mathsf{j},\mathsf{j}') \ * \ \mathsf{dlist}\,\beta\,(\mathsf{j},\mathsf{j}',\mathsf{k},\mathsf{k}')$$

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

Emptyness Conditions

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

Deleting an Element from a Doubly-Linked List



$\{\exists \mathsf{j}, \mathsf{l}.\ \mathsf{dlist}\,\alpha\,(\mathsf{i}, \mathbf{nil}, \mathsf{k}, \mathsf{j})\ *\ \mathsf{k} \mapsto \mathsf{b}, \mathsf{l}, \mathsf{j}\ *\ \mathsf{dlist}\,\beta\,(\mathsf{l}, \mathsf{k}, \mathbf{nil}, \mathsf{m})\}$

$\mathsf{l} := [\mathsf{k}+1]\ ;\ \mathsf{j} := [\mathsf{k}+2]\ ;$

$\{\mathsf{dlist}\,\alpha\,(\mathsf{i}, \mathbf{nil}, \mathsf{k}, \mathsf{j})\ *\ \mathsf{k} \mapsto \mathsf{b}, \mathsf{l}, \mathsf{j}\ *\ \mathsf{dlist}\,\beta\,(\mathsf{l}, \mathsf{k}, \mathbf{nil}, \mathsf{m})\}$

dispose k ; **dispose** $\mathsf{k}+1$; **dispose** $\mathsf{k}+2$;

$\{\mathsf{dlist}\,\alpha\,(\mathsf{i}, \mathbf{nil}, \mathsf{k}, \mathsf{j})\ *\ \mathsf{dlist}\,\beta\,(\mathsf{l}, \mathsf{k}, \mathbf{nil}, \mathsf{m})\}$

if $\mathsf{j} = \mathbf{nil}$ **then**

$\{\mathsf{dlist}\,\beta\,(\mathsf{l}, \mathsf{k}, \mathbf{nil}, \mathsf{m}) \wedge \mathsf{i} = \mathsf{k} \wedge \mathbf{nil} = \mathsf{j} \wedge \alpha = \epsilon\}$

$\mathsf{i} := \mathsf{l}$

$\{\mathsf{dlist}\,\beta\,(\mathsf{l}, \mathsf{k}, \mathbf{nil}, \mathsf{m}) \wedge \mathsf{i} = \mathsf{l} \wedge \mathbf{nil} = \mathsf{j} \wedge \alpha = \epsilon\}$

else

$\{\exists \alpha', \mathsf{a}, \mathsf{n}.\ (\mathsf{dlist}\,\alpha'\,(\mathsf{i}, \mathbf{nil}, \mathsf{j}, \mathsf{n})\ *\ \mathsf{j} \mapsto \mathsf{a}, \mathsf{k}, \mathsf{n}$

$*\ \mathsf{dlist}\,\beta\,(\mathsf{l}, \mathsf{k}, \mathbf{nil}, \mathsf{m})) \wedge \alpha = \alpha' \cdot \mathsf{a}\}$

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

$\{\exists \alpha', \mathsf{a}, \mathsf{n}.\ (\mathsf{dlist}\,\alpha'\,(\mathsf{i}, \mathbf{nil}, \mathsf{j}, \mathsf{n})\ *\ \mathsf{j} \mapsto \mathsf{a}, \mathsf{l}, \mathsf{n}$

$*\ \mathsf{dlist}\,\beta\,(\mathsf{l}, \mathsf{k}, \mathbf{nil}, \mathsf{m})) \wedge \alpha = \alpha' \cdot \mathsf{a}\}$

$\{\mathsf{dlist}\,\alpha\,(\mathsf{i}, \mathbf{nil}, \mathsf{l}, \mathsf{j})\ *\ \mathsf{dlist}\,\beta\,(\mathsf{l}, \mathsf{k}, \mathbf{nil}, \mathsf{m})\}$

$\vdots$

Deleting from a Doubly-Linked List (continued)

$$
\begin{array}{l}
\quad \vdots \\
\{\mathsf{dlist}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{l}, \mathsf{j})\ *\ \mathsf{dlist}\ \beta\ (\mathsf{l}, \mathsf{k}, \mathbf{nil}, \mathsf{m})\} \\
\mathbf{if}\ \mathsf{l} = \mathbf{nil}\ \mathbf{then} \\
\quad \{\mathsf{dlist}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{l}, \mathsf{j}) \wedge \mathsf{l} = \mathbf{nil} \wedge \mathsf{k} = \mathsf{m} \wedge \beta = \epsilon\} \\
\quad \mathsf{m} := \mathsf{j} \\
\quad \{\mathsf{dlist}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{l}, \mathsf{j}) \wedge \mathsf{l} = \mathbf{nil} \wedge \mathsf{j} = \mathsf{m} \wedge \beta = \epsilon\} \\
\mathbf{else} \\
\quad \{\exists \mathsf{a}, \beta', \mathsf{n}.\ (\mathsf{dlist}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{l}, \mathsf{j})\ *\ \mathsf{l} \mapsto \mathsf{a}, \mathsf{n}, \mathsf{k} \\
\qquad *\ \mathsf{dlist}\ \beta'\ (\mathsf{n}, \mathsf{l}, \mathbf{nil}, \mathsf{m})) \wedge \beta = \mathsf{a}{\cdot}\beta'\} \\
\quad [\mathsf{l} + 2] := \mathsf{j} \\
\quad \{\exists \mathsf{a}, \beta', \mathsf{n}.\ (\mathsf{dlist}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{l}, \mathsf{j})\ *\ \mathsf{l} \mapsto \mathsf{a}, \mathsf{n}, \mathsf{j} \\
\qquad *\ \mathsf{dlist}\ \beta'\ (\mathsf{n}, \mathsf{l}, \mathbf{nil}, \mathsf{m})) \wedge \beta = \mathsf{a}{\cdot}\beta'\} \\
\{\mathsf{dlist}\ \alpha\ (\mathsf{i}, \mathbf{nil}, \mathsf{l}, \mathsf{j})\ *\ \mathsf{dlist}\ \beta\ (\mathsf{l}, \mathsf{j}, \mathbf{nil}, \mathsf{m})\} \\
\{\mathsf{dlist}\ \alpha{\cdot}\beta\ (\mathsf{i}, \mathbf{nil}, \mathbf{nil}, \mathsf{m})\}
\end{array}
$$

An Alternative Intuitionistic Logic

- We drop the assertion **emptyheap**.
- We drop the assertion $\mapsto$, and make $\hookrightarrow$ primitive.
- We alter the definitions of $\neg$, $\Rightarrow$, and $\Leftrightarrow$.

$[\![e \hookrightarrow e']\!]_{\text{assert}}(s,h)$

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

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

iff $\forall h'.\ h \subseteq h' \Rightarrow \neg[\![p]\!]_{\text{assert}}(s,h')$,

$[\![p_0 \Rightarrow p_1]\!]_{\text{assert}}(s,h)$

iff $\forall h'.\ h \subseteq h' \Rightarrow [\![p_0]\!]_{\text{assert}}(s,h') \Rightarrow [\![p_1]\!]_{\text{assert}}(s,h')$,

$[\![p_0 \Leftrightarrow p_1]\!]_{\text{assert}}(s,h)$

iff $[\![p_0 \Rightarrow p_1]\!]_{\text{assert}}(s,h)$ and $[\![p_1 \Rightarrow p_0]\!]_{\text{assert}}(s,h)$.

The Advantage: Monotonicity

$h \subseteq h'$ and $[\![p]\!]_{\text{assert}}(s,h)$ implies $[\![p]\!]_{\text{assert}}(s,h')$.

The Disadvantage: Failure of the Law of the Excluded Middle

$\mathsf{x} \hookrightarrow 13 \vee (\neg\, \mathsf{x} \hookrightarrow 13)$ fails when $h = [\mathsf{y} : 14]$.

A Problematic Specification

$$\{\neg(\exists \mathsf{x}.\ \mathsf{x} \to 1, 2)\}$$
$$\mathsf{y} := \mathbf{cons}(1, 2)$$
$$\{(\neg(\exists \mathsf{x}.\ \mathsf{x} \to 1, 2)) \ * \ \mathsf{y} \to 1, 2\}.$$

Translating from Intuitionistic to Classical Semantics

The translation $(-)^o$, from intuitionistic to classical assertions, sends

$$e \hookrightarrow e' \ \text{ to } \ e \mapsto e' \ * \ \mathbf{true}$$
$$\neg\, p \ \text{ to } \ \mathbf{true} \mathrel{-\!\!*} (\neg\, p^o)$$
$$p \Rightarrow q \ \text{ to } \ \mathbf{true} \mathrel{-\!\!*} (p^o \Rightarrow q^o)$$
$$p \Leftrightarrow q \ \text{ to } \ \mathbf{true} \mathrel{-\!\!*} (p^o \Leftrightarrow q^o),$$

and everything else (inductively) to itself.

Future Directions

- Trees and other abstract data types
- Directed graphs
- Weakest preconditions, dynamic logic, and reference calculi
- Array allocation
- Assertions versus types
- Garbage collection

$$
\begin{array}{l}
\{\mathbf{emptyheap}\} \\
\mathsf{x} := \mathbf{cons}_2(1,2)\ ; \\
\{\mathsf{x} \rightarrow 1,2\} \\
\mathsf{x} := 3 \\
\{(\exists \mathsf{x}.\ \mathsf{x} \rightarrow 1,2) \wedge \mathsf{x} = 3\}
\end{array}
$$

- Higher-order programming (code pointers)

Answer 1.1

1. $$\exists \mathsf{x}.\ \mathsf{x} > 0 \wedge \mathsf{x} < 2$$

2. $$\forall \mathsf{x}.\ \forall \mathsf{y}.\ \mathsf{x} > 0 \wedge \mathsf{x} < 2 \wedge \mathsf{y} > 0 \wedge \mathsf{y} < 2 \Rightarrow \mathsf{x} = \mathsf{y}$$

3. $$\exists \mathsf{x}\ \exists \mathsf{y}.\ \mathsf{x} > 0 \wedge \mathsf{x} < 3 \wedge \mathsf{y} > 0 \wedge \mathsf{y} < 3 \wedge \mathsf{x} \neq \mathsf{y}$$

4. $$\forall \mathsf{x}.\ \forall \mathsf{y}.\ \forall \mathsf{z}.\ \mathsf{x} > 0 \wedge \mathsf{x} < 3 \wedge \mathsf{y} > 0 \wedge \mathsf{y} < 3 \wedge \mathsf{z} > 0 \wedge \mathsf{z} < 3$$
$$\Rightarrow (\mathsf{x} = \mathsf{y} \vee \mathsf{y} = \mathsf{z} \vee \mathsf{z} = \mathsf{x})$$

1. $$\exists \mathsf{x}.\ \mathsf{a} \times \mathsf{x} = \mathsf{b}$$

2. $$(\exists \mathsf{x}.\ \mathsf{a} \times \mathsf{x} = \mathsf{b}) \wedge (\exists \mathsf{y}.\ \mathsf{a} \times \mathsf{y} = \mathsf{c})$$

3. $$(\exists \mathsf{x}.\ \mathsf{a} \times \mathsf{x} = \mathsf{b}) \wedge (\exists \mathsf{y}.\ \mathsf{a} \times \mathsf{y} = \mathsf{c})$$
$$\wedge\ ((\exists \mathsf{x}.\ \mathsf{a}' \times \mathsf{x} = \mathsf{b}) \wedge \forall \mathsf{a}'.\ (\exists \mathsf{y}.\ \mathsf{a}' \times \mathsf{y} = \mathsf{c})) \Rightarrow \mathsf{a} \geq \mathsf{a}'$$

4. $$\forall \mathsf{d}.\ 1 < \mathsf{d} \wedge \mathsf{d} < \mathsf{p} \Rightarrow \neg\, (\exists \mathsf{x}.\ \mathsf{d} \times \mathsf{x} = \mathsf{p})$$

Answer 1.2

1. $$\langle \text{exp} \rangle ::= \textbf{sum}\ \langle \text{var} \rangle : \langle \text{exp} \rangle\ \textbf{to}\ \langle \text{exp} \rangle .\ \langle \text{exp} \rangle$$

2. $$[\![\textbf{sum}\ v : e_0\ \textbf{to}\ e_1.\ e_2]\!]_{\text{exp}}\sigma = \sum_{i=[\![e_0]\!]_{\text{exp}}\sigma}^{[\![e_1]\!]_{\text{exp}}\sigma} [\![e_2]\!]_{\text{exp}}[\,\sigma \mid v{:}\, i\,]$$

3. The first occurrence of v in $\textbf{sum}\ v : e_0\ \textbf{to}\ e_1.\ e_2$ is a binder whose scope is e_2.

4. $$e_0 > e_1 \Rightarrow \textbf{sum}\ v : e_0\ \textbf{to}\ e_1.\ e_2 = 0$$

   $$e_0 \leq e_1 \Rightarrow \textbf{sum}\ v : e_0\ \textbf{to}\ e_1.\ e_2$$
   $$= (\textbf{sum}\ v : e_0\ \textbf{to}\ e_1 - 1.\ e_2) + (e_2/v \rightarrow e_1)$$

   $$e_0 = e_1 \Rightarrow \textbf{sum}\ v : e_0\ \textbf{to}\ e_1.\ e_2 = (e_2/v \rightarrow e_0)$$

   $$e_0 \leq e_1 \wedge e_1 \leq e_2 + 1 \Rightarrow \textbf{sum}\ v : e_0\ \textbf{to}\ e_2.\ e_3$$
   $$= (\textbf{sum}\ v : e_0\ \textbf{to}\ e_1 - 1.\ e_3) + (\textbf{sum}\ v : e_1\ \textbf{to}\ e_2.\ e_3)$$

Answer 2.1

$\langle \textbf{while } \mathsf{x} \neq 0 \textbf{ do if } \mathsf{x} = 1 \textbf{ then skip else } \mathsf{x} := \mathsf{x} - 2, [\mathsf{x} : 2] \rangle \rightsquigarrow$

$\langle \textbf{if } \mathsf{x} = 1 \textbf{ then skip else } \mathsf{x} := \mathsf{x} - 2 \; ;$

$\quad \textbf{while } \mathsf{x} \neq 0 \textbf{ do if } \mathsf{x} = 1 \textbf{ then skip else } \mathsf{x} := \mathsf{x} - 2, [\mathsf{x} : 2] \rangle \rightsquigarrow$

$\langle \mathsf{x} := \mathsf{x} - 2 \; ;$

$\quad \textbf{while } \mathsf{x} \neq 0 \textbf{ do if } \mathsf{x} = 1 \textbf{ then skip else } \mathsf{x} := \mathsf{x} - 2, [\mathsf{x} : 2] \rangle \rightsquigarrow$

$\langle \textbf{while } \mathsf{x} \neq 0 \textbf{ do if } \mathsf{x} = 1 \textbf{ then skip else } \mathsf{x} := \mathsf{x} - 2, [\mathsf{x} : 0] \rangle \rightsquigarrow$

$[\mathsf{x} : 0]$

By the second rule for conditional commands:

$\langle \textbf{if } \mathsf{x} = 1 \textbf{ then skip else } \mathsf{x} := \mathsf{x} - 2, [\mathsf{x} : 2] \rangle \rightsquigarrow \langle \mathsf{x} := \mathsf{x} - 2, [\mathsf{x} : 2] \rangle$

Then, by the second rule for sequencing:

$\langle \textbf{if } \mathsf{x} = 1 \textbf{ then skip else } \mathsf{x} := \mathsf{x} - 2 \; ;$

$\quad \textbf{while } \mathsf{x} \neq 0 \textbf{ do if } \mathsf{x} = 1 \textbf{ then skip else } \mathsf{x} := \mathsf{x} - 2, [\mathsf{x} : 2] \rangle \rightsquigarrow$

$\langle \mathsf{x} := \mathsf{x} - 2 \; ;$

$\quad \textbf{while } \mathsf{x} \neq 0 \textbf{ do if } \mathsf{x} = 1 \textbf{ then skip else } \mathsf{x} := \mathsf{x} - 2, [\mathsf{x} : 2] \rangle$

Answer 2.2: The **repeat** Command

$$\langle \text{comm} \rangle ::= \textbf{repeat } \langle \text{comm} \rangle \textbf{ until } \langle \text{boolexp} \rangle$$

Semantics

$$\frac{}{\langle \textbf{repeat } c \textbf{ until } b, \sigma \rangle \rightsquigarrow \langle c \,;\, \textbf{while } \neg\, b \textbf{ do } c, \sigma \rangle}$$

or

$$\frac{\langle c, \sigma \rangle \rightsquigarrow \langle c', \sigma' \rangle}{\langle \textbf{repeat } c \textbf{ until } b, \sigma \rangle \rightsquigarrow \langle c' \,;\, \textbf{while } \neg\, b \textbf{ do } c, \sigma' \rangle}$$

$$\frac{\langle c, \sigma \rangle \rightsquigarrow \sigma'}{\langle \textbf{repeat } c \textbf{ until } b, \sigma \rangle \rightsquigarrow \langle \textbf{while } \neg\, b \textbf{ do } c, \sigma' \rangle}$$

or

$$\frac{}{\langle \textbf{repeat } c \textbf{ until } b, \sigma \rangle \rightsquigarrow \langle c \,;\, \textbf{if } b \textbf{ then skip else repeat } c \textbf{ until } b, \sigma \rangle}$$

or

$$\frac{\langle c, \sigma \rangle \rightsquigarrow \langle c', \sigma' \rangle}{\langle \textbf{repeat } c \textbf{ until } b, \sigma \rangle \rightsquigarrow \langle c' \,;\, \textbf{if } b \textbf{ then skip else repeat } c \textbf{ until } b, \sigma' \rangle}$$

$$\frac{\langle c, \sigma \rangle \rightsquigarrow \sigma'}{\langle \textbf{repeat } c \textbf{ until } b, \sigma \rangle \rightsquigarrow \langle \textbf{if } b \textbf{ then skip else repeat } c \textbf{ until } b, \sigma' \rangle}$$

Answer 3.1

$$[\, \mathsf{y} = \mathsf{x}^2 \,]\ \mathsf{x} := \mathsf{x} + 1\ ;\ \mathsf{y} := \mathsf{y} + 2 \times \mathsf{x} - 1\ [\, \mathsf{y} = \mathsf{x}^2 \,]$$

$$[\mathsf{y} = \mathsf{z} \wedge \mathsf{x} = \mathsf{w}]$$
$$\mathsf{x} := \mathsf{x} + \mathsf{y}\ ;\ \mathsf{y} := \mathsf{x} - \mathsf{y}\ ;\ \mathsf{x} := \mathsf{x} - \mathsf{y}$$
$$[\mathsf{x} = \mathsf{z} \wedge \mathsf{y} = \mathsf{w}]$$

$$[\mathsf{y} = \mathsf{x} \times \mathsf{a} \wedge \mathsf{a} \leq \mathsf{b} \vee \mathsf{y} = \mathsf{x} \times \mathsf{b} \wedge \mathsf{a} \geq \mathsf{b}]$$
$$\textbf{while}\ \mathsf{a} < \mathsf{b}\ \textbf{do}\ (\mathsf{a} := \mathsf{a} + 1\ ;\ \mathsf{y} := \mathsf{x} + \mathsf{y})$$
$$[\mathsf{y} = \mathsf{x} \times \mathsf{b}]$$

$$\{\textbf{true}\}\ \textbf{while true do skip}\ \{\textbf{false}\}$$

$$[\,\textbf{false}\,]\ \textbf{while true do skip}\ [\,\textbf{false}\,]$$

Answer 3.2: Logarithmic-Time Multiplication

$$
\begin{array}{l}
[\mathsf{y} \geq 0] \\
\mathsf{z} := 0 \,;\, \mathsf{w} := \mathsf{x} \,;\, \mathsf{k} := \mathsf{y} \,; \\
[\mathsf{z} + \mathsf{w} \times \mathsf{k} = \mathsf{x} \times \mathsf{y} \wedge \mathsf{k} \geq 0] \\
\textbf{while } [\textbf{vrnt}\colon \mathsf{k}]\, \mathsf{k} \neq 0 \textbf{ do} \\
\quad \textbf{if even } \mathsf{k} \textbf{ then } (\mathsf{k} := \mathsf{k} \div 2 \,;\, \mathsf{w} := \mathsf{w} + \mathsf{w}) \\
\qquad \textbf{else } (\mathsf{k} := \mathsf{k} - 1 \,;\, \mathsf{z} := \mathsf{z} + \mathsf{w}) \\
[\mathsf{z} = \mathsf{x} \times \mathsf{y}]
\end{array}
$$

The verification conditions are:

$$\mathsf{y} \geq 0 \Rightarrow (0 + \mathsf{x} \times \mathsf{y} = \mathsf{x} \times \mathsf{y} \wedge \mathsf{y} \geq 0)$$

$$
\begin{array}{l}
(\mathsf{z} + \mathsf{w} \times \mathsf{k} = \mathsf{x} \times \mathsf{y} \wedge \mathsf{k} \geq 0 \wedge \mathsf{k} \neq 0 \wedge \mathsf{k} = \mathsf{k}_0 \wedge \textbf{even}\, \mathsf{k}) \Rightarrow \\
\quad (\mathsf{z} + (\mathsf{w} + \mathsf{w}) \times (\mathsf{k} \div 2) = \mathsf{x} \times \mathsf{y} \wedge \mathsf{k} \div 2 \geq 0 \wedge \mathsf{k} \div 2 < \mathsf{k}_0)
\end{array}
$$

$$
\begin{array}{l}
(\mathsf{z} + \mathsf{w} \times \mathsf{k} = \mathsf{x} \times \mathsf{y} \wedge \mathsf{k} \geq 0 \wedge \mathsf{k} \neq 0 \wedge \mathsf{k} = \mathsf{k}_0 \wedge \neg\, \textbf{even}\, \mathsf{k}) \Rightarrow \\
\quad ((\mathsf{z} + \mathsf{w}) + \mathsf{w} \times (\mathsf{k} - 1) = \mathsf{x} \times \mathsf{y} \wedge \mathsf{k} - 1 \geq 0 \wedge \mathsf{k} - 1 < \mathsf{k}_0)
\end{array}
$$

$$(\mathsf{z} + \mathsf{w} \times \mathsf{k} = \mathsf{x} \times \mathsf{y} \wedge \mathsf{k} \geq 0 \wedge \mathsf{k} \neq 0) \Rightarrow \mathsf{k} \geq 0$$

$$(\mathsf{z} + \mathsf{w} \times \mathsf{k} = \mathsf{x} \times \mathsf{y} \wedge \mathsf{k} \geq 0 \wedge \neg\ \mathsf{k} \neq 0) \Rightarrow \mathsf{z} = \mathsf{x} \times \mathsf{y}$$

Answer 3.3: Logarithmic-Time Division

$[\mathsf{x} \geq 0 \wedge \mathsf{y} > 0]$

$\mathsf{n} := 0 \,;\, \mathsf{z} := \mathsf{y} \,;$

$[\mathsf{z} = \mathsf{y} \times 2^{\mathsf{n}} \wedge \mathsf{n} \geq 0 \wedge \mathsf{x} \geq 0 \wedge \mathsf{y} > 0]$

while $[\mathbf{vrnt}\colon \mathsf{x} - \mathsf{z}]\, \mathsf{z} \leq \mathsf{x}$ **do** $(\mathsf{n} := \mathsf{n} + 1 \,;\, \mathsf{z} = \mathsf{z} \times 2) \,;$

$\mathsf{q} := 0 \,;\, \mathsf{r} := \mathsf{x} \,;$

$[\mathsf{x} = \mathsf{q} \times \mathsf{z} + \mathsf{r} \wedge 0 \leq \mathsf{r} < \mathsf{z} \wedge \mathsf{z} = \mathsf{y} \times 2^{\mathsf{n}} \wedge \mathsf{n} \geq 0]$

while $[\mathbf{vrnt}\colon \mathsf{n}]\, \mathsf{n} \neq 0$ **do**

$\Big(\mathsf{n} := \mathsf{n} - 1 \,;\, \mathsf{z} := \mathsf{z} \div 2 \,;\, \mathsf{q} := \mathsf{q} \times 2 \,;$

$[\mathsf{x} = \mathsf{q} \times \mathsf{z} + \mathsf{r} \wedge 0 \leq \mathsf{r} < \mathsf{z} \times 2 \wedge \mathsf{z} = \mathsf{y} \times 2^{\mathsf{n}} \wedge \mathsf{n} \geq 0 \wedge \mathsf{n} < \mathsf{n}_0]$

if $\mathsf{z} \leq \mathsf{r}$ **then** $\mathsf{q} := \mathsf{q} + 1 \,;\, \mathsf{r} := \mathsf{r} - \mathsf{z}$ **else skip**$\Big)$

$[\mathsf{x} = \mathsf{q} \times \mathsf{y} + \mathsf{r} \wedge 0 \leq \mathsf{r} < \mathsf{y}]$

Let

$$I_1 \stackrel{\text{def}}{=} \mathsf{z} = \mathsf{y} \times 2^{\mathsf{n}} \wedge \mathsf{n} \geq 0 \wedge \mathsf{x} \geq 0 \wedge \mathsf{y} > 0$$

$$I_2 \stackrel{\text{def}}{=} \mathsf{x} = \mathsf{q} \times \mathsf{z} + \mathsf{r} \wedge 0 \leq \mathsf{r} < \mathsf{z} \wedge \mathsf{z} = \mathsf{y} \times 2^{\mathsf{n}} \wedge \mathsf{n} \geq 0.$$

Answer 3.3: Logarithmic-Time Division (continued)

Then the verification conditions are

$$(\mathsf{x} \geq 0 \wedge \mathsf{y} > 0) \Rightarrow (\mathsf{y} = \mathsf{y} \times 2^0 \wedge 0 \geq 0 \wedge \mathsf{x} \geq 0 \wedge \mathsf{y} > 0)$$

$$(I_1 \wedge \mathsf{z} \leq \mathsf{x} \wedge \mathsf{x} - \mathsf{z} = \mathsf{k}_0) \Rightarrow$$
$$(\mathsf{z} \times 2 = \mathsf{y} \times 2^{\mathsf{n}+1} \wedge \mathsf{n}+1 \geq 0 \wedge \mathsf{x} \geq 0 \wedge \mathsf{y} > 0 \wedge \mathsf{x} - \mathsf{z} \times 2 < \mathsf{k}_0)$$

$$(I_1 \wedge \mathsf{z} \leq \mathsf{x}) \Rightarrow \mathsf{x} - \mathsf{z} \geq 0$$

$$(I_1 \wedge \neg\ \mathsf{z} \leq \mathsf{x}) \Rightarrow$$
$$(\mathsf{x} = 0 \times \mathsf{z} + \mathsf{x} \wedge 0 \leq \mathsf{x} < \mathsf{z} \wedge \mathsf{z} = \mathsf{y} \times 2^{\mathsf{n}} \wedge \mathsf{n} \geq 0)$$

$$(I_2 \wedge \mathsf{n} \neq 0 \wedge \mathsf{n} = \mathsf{n}_0) \Rightarrow$$
$$\big(\mathsf{x} = (\mathsf{q} \times 2) \times (\mathsf{z} \div 2) + \mathsf{r} \wedge 0 \leq \mathsf{r} < (\mathsf{z} \div 2) \times 2 \wedge$$
$$\mathsf{z} \div 2 = \mathsf{y} \times 2^{\mathsf{n}-1} \wedge \mathsf{n} - 1 \geq 0 \wedge \mathsf{n} - 1 < \mathsf{n}_0\big)$$

$$(\mathsf{x} = \mathsf{q} \times \mathsf{z} + \mathsf{r} \wedge 0 \leq \mathsf{r} < \mathsf{z} \times 2 \wedge$$
$$\mathsf{z} = \mathsf{y} \times 2^{\mathsf{n}} \wedge \mathsf{n} \geq 0 \wedge \mathsf{n} < \mathsf{n}_0 \wedge \mathsf{z} \leq \mathsf{r}) \Rightarrow$$
$$\big(\mathsf{x} = (\mathsf{q}+1) \times \mathsf{z} + (\mathsf{r} - \mathsf{z}) \wedge 0 \leq \mathsf{r} - \mathsf{z} < \mathsf{z} \wedge$$
$$\mathsf{z} = \mathsf{y} \times 2^{\mathsf{n}} \wedge \mathsf{n} \geq 0 \wedge \mathsf{n} < \mathsf{n}_0\big)$$

$$(\mathsf{x} = \mathsf{q} \times \mathsf{z} + \mathsf{r} \wedge 0 \leq \mathsf{r} < \mathsf{z} \times 2 \wedge$$
$$\mathsf{z} = \mathsf{y} \times 2^{\mathsf{n}} \wedge \mathsf{n} \geq 0 \wedge \mathsf{n} < \mathsf{n}_0 \wedge \neg\ \mathsf{z} \leq \mathsf{r}) \Rightarrow$$
$$(I_2 \wedge \mathsf{n} < \mathsf{n}_0)$$

$$(I_2 \wedge \mathsf{n} \neq 0) \Rightarrow \mathsf{n} \geq 0$$

$$(I_2 \wedge \neg\ \mathsf{n} \neq 0) \Rightarrow (\mathsf{x} = \mathsf{q} \times \mathsf{y} + \mathsf{r} \wedge 0 \leq \mathsf{r} < \mathsf{y})$$

Answers 4.1

$$p_1 \wedge p_2 \not\Rightarrow p_1 \ * \ p_2$$

$$p_1 : \mathsf{x} \mapsto 1$$
$$p_2 : \mathsf{x} \mapsto 1$$
$$[s\mathsf{x} : 1]$$

$$p_1 \ * \ p_2 \not\Rightarrow p_1 \wedge p_2$$

$$p_1 : \mathsf{x} \mapsto 1$$
$$p_2 : \mathbf{true}$$
$$[s\mathsf{x} : 1, s\mathsf{y} : 2]$$

$$(p_1 \ * \ q) \wedge (p_2 \ * \ q) \not\Rightarrow (p_1 \wedge p_2) \ * \ q$$

$$p_1 : \mathsf{x} \mapsto 1$$
$$p_2 : \mathsf{y} \mapsto 2$$
$$q : \mathsf{x} \mapsto 1 \vee \mathsf{y} \mapsto 2$$
$$[s\mathsf{x} : 1, s\mathsf{y} : 2]$$

$$(p_1 \vee q) \ * \ (p_2 \vee q) \not\Rightarrow (p_1 \ * \ p_2) \vee q$$

$$p_1 : \mathsf{x} \mapsto 1$$
$$p_2 : \mathsf{x} \mapsto 1$$
$$q : \mathsf{y} \mapsto 2$$
$$[s\mathsf{x} : 1, s\mathsf{y} : 2]$$

$$(p_1 \ * \ p_2) \vee q \not\Rightarrow (p_1 \vee q) \ * \ (p_2 \vee q)$$

$$p_1 : \mathsf{x} \mapsto 1$$
$$p_2 : \mathsf{x} \mapsto 1$$
$$q : \mathsf{x} \mapsto 1$$
$$[s\mathsf{x} : 1]$$

Answers 4.1 (continued)

$(p_1 \wedge q) \ * \ (p_2 \wedge q) \not\Rightarrow (p_1 \ * \ p_2) \wedge q$

$p_1 : \mathsf{x} \mapsto 1$

$p_2 : \mathsf{y} \mapsto 2$

$q : \mathsf{x} \mapsto 1 \vee \mathsf{y} \mapsto 2$

$[s\mathsf{x} : 1, s\mathsf{y} : 2]$

$(p_1 \ * \ p_2) \wedge q \not\Rightarrow (p_1 \wedge q) \ * \ (p_2 \wedge q)$

$p_1 : \mathsf{x} \mapsto 1$

$p_2 : \mathsf{y} \mapsto 2$

$q : \mathsf{x} \mapsto 1 \ * \ \mathsf{y} \mapsto 2$

$[s\mathsf{x} : 1, s\mathsf{y} : 2]$

$(p_1 \Rightarrow p_2) \not\Rightarrow (p_1 \ * \ q) \Rightarrow (p_2 \ * \ q)$

$p_1 : \mathsf{x} \mapsto 1 \vee (\mathsf{x} \mapsto 1 \ * \ \mathsf{y} \mapsto 2)$

$p_2 : \mathbf{true} \ * \ \mathsf{y} \mapsto 2$

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

$[s\mathsf{x} : 1, s\mathsf{y} : 2]$

$\forall x. \ (p_1 \ * \ p_2) \not\Rightarrow (\forall x. \ p_1) \ * \ p_2$ when x not free in p_2

$p_1 : \mathbf{if} \ \mathsf{x} = 0 \ \mathbf{then} \ \mathsf{y} \mapsto 1 \ \mathbf{else} \ \mathsf{z} \mapsto 2$

$p_2 : \mathsf{y} \mapsto 1 \vee \mathsf{z} \mapsto 2$

$[s\mathsf{y} : 1, s\mathsf{z} : 2]$

Answers 4.1 (continued)

$(p_1 \mathrel{-\!\!*} p_2) \not\Rightarrow (p_1 \Rightarrow p_2)$

$p_1 : \mathsf{y} \mapsto 1$

$p_2 : \mathsf{x} \neq \mathsf{y}$

$[s\mathsf{y} : 1]$

$(p_1 \Rightarrow p_2) \not\Rightarrow (p_1 \mathrel{-\!\!*} p_2)$

$p_1 : \mathbf{true}$

$p_2 : \mathsf{x} \mapsto 1$

$[s\mathsf{x} : 1]$

or $p_1 : \mathsf{x} \mapsto 1$

$p_2 : \mathbf{false}$

$[\,]$

Answer 5.1

1.

$$\exists \alpha, \beta, \delta.\ (\mathbf{list}\ \alpha\ (\mathsf{i}, \mathsf{j}_1)\ *\ \mathbf{list}\ \beta\ (\mathsf{j}_1, \mathsf{j}_2)\ *\ \mathbf{list}\ \delta\ (\mathsf{j}_2, \mathsf{k})) \\ \wedge\ \gamma = \alpha \cdot \beta \cdot \delta$$

2.

$$(\exists \alpha, \beta.\ (\mathbf{list}\ \alpha\ (\mathsf{i}, \mathsf{j}_1)\ *\ \mathbf{list}\ \beta\ (\mathsf{j}_1, \mathsf{k})) \wedge \gamma = \alpha \cdot \beta) \\ \wedge\ (\exists \alpha, \beta.\ (\mathbf{list}\ \alpha\ (\mathsf{i}, \mathsf{j}_2)\ *\ \mathbf{list}\ \beta\ (\mathsf{j}_2, \mathsf{k})) \wedge \gamma = \alpha \cdot \beta)$$

3.

$$\exists \alpha, \beta, \delta.\ (\mathbf{list}\ \alpha\ (\mathsf{i}, \mathsf{j}_1)\ *\ \mathbf{list}\ \beta\ (\mathsf{j}_1, \mathsf{j}_2)\ *\ \mathbf{list}\ \delta\ (\mathsf{j}_2, \mathsf{k})) \\ \wedge\ \gamma = \alpha \cdot \beta \cdot \delta$$

$$\Rightarrow$$

$$\Big(\exists \alpha, \beta, \delta.\ (\mathbf{list}\ \alpha\ (\mathsf{i}, \mathsf{j}_1)\ *\ (\mathbf{list}\ \beta\ (\mathsf{j}_1, \mathsf{j}_2)\ *\ \mathbf{list}\ \delta\ (\mathsf{j}_2, \mathsf{k}))) \\ \wedge\ \gamma = \alpha \cdot \beta \cdot \delta\Big) \\ \wedge\ \Big(\exists \alpha, \beta, \delta.\ ((\mathbf{list}\ \alpha\ (\mathsf{i}, \mathsf{j}_1)\ *\ \mathbf{list}\ \beta\ (\mathsf{j}_1, \mathsf{j}_2))\ *\ \mathbf{list}\ \delta\ (\mathsf{j}_2, \mathsf{k})) \\ \wedge\ \gamma = \alpha \cdot \beta \cdot \delta\Big)$$

$$\Rightarrow$$

$$\Big(\exists \alpha, \beta, \delta.\ (\mathbf{list}\ \alpha\ (\mathsf{i}, \mathsf{j}_1)\ *\ \mathbf{list}\ \beta \cdot \delta\ (\mathsf{j}_1, \mathsf{k})) \wedge \gamma = \alpha \cdot \beta \cdot \delta\Big) \\ \wedge\ \Big(\exists \alpha, \beta, \delta.\ (\mathbf{list}\ \alpha \cdot \beta\ (\mathsf{i}, \mathsf{j}_2)\ *\ \mathbf{list}\ \delta\ (\mathsf{j}_2, \mathsf{k})) \wedge \gamma = \alpha \cdot \beta \cdot \delta\Big)$$

$$\Rightarrow$$

$$\Big(\exists \alpha, \beta.\ (\mathbf{list}\ \alpha\ (\mathsf{i}, \mathsf{j}_1)\ *\ \mathbf{list}\ \beta\ (\mathsf{j}_1, \mathsf{k})) \wedge \gamma = \alpha \cdot \beta\Big) \\ \wedge\ \Big(\exists \alpha, \beta.\ (\mathbf{list}\ \alpha\ (\mathsf{i}, \mathsf{j}_2)\ *\ \mathbf{list}\ \beta\ (\mathsf{j}_2, \mathsf{k})) \wedge \gamma = \alpha \cdot \beta\Big)$$

Answer 5.2.1

There are two cases, depending upon whether α is empty:

$$\begin{aligned}
&\mathbf{brlist}\ \epsilon\,(\mathsf{i},\mathsf{j},\mathsf{k}) \\
&\qquad \Rightarrow \mathbf{emptyheap} \wedge \mathsf{i} = \mathsf{k} \\
&\qquad \Rightarrow \mathbf{list}\ \epsilon\,(\mathsf{i},\mathsf{k}) \\
&\mathbf{brlist}\ \alpha'{\cdot}\mathsf{a}\,(\mathsf{i},\mathsf{j},\mathsf{k}) \\
&\qquad \Rightarrow \mathbf{list}\ \alpha'\,(\mathsf{i},\mathsf{j})\ *\ \mathsf{j} \mapsto \mathsf{a},\mathsf{k} \\
&\qquad \Rightarrow \exists \mathsf{j}.\ \mathbf{list}\ \alpha'\,(\mathsf{i},\mathsf{j})\ *\ \mathsf{j} \mapsto \mathsf{a},\mathsf{k} \\
&\qquad \Rightarrow \mathbf{list}\ \alpha'{\cdot}\mathsf{a}\,(\mathsf{i},\mathsf{k})
\end{aligned}$$

Answer 5.2.2

$\{\mathbf{brlist}\ \alpha\,(\mathsf{i},\mathsf{j},\mathsf{k})\}$

if $\mathsf{j} = \mathbf{nil}$ **then**

$\{\mathbf{brlist}\ \epsilon\,(\mathsf{i},\mathsf{j},\mathsf{k}) \wedge \alpha = \epsilon\}$

$\{\mathsf{j} = \mathbf{nil} \wedge \mathbf{emptyheap} \wedge \alpha = \epsilon\}$

$\mathsf{i} := \mathsf{l}$

$\{\mathsf{j} = \mathbf{nil} \wedge \mathbf{emptyheap} \wedge \mathsf{i} = \mathsf{l} \wedge \alpha = \epsilon\}$

else

$\{\exists \alpha', \mathsf{a}.\ \mathbf{brlist}\ \alpha' \cdot \mathsf{a}\,(\mathsf{i},\mathsf{j},\mathsf{k}) \wedge \alpha = \alpha' \cdot \mathsf{a}\}$

$\{\exists \alpha', \mathsf{a}.\ (\mathbf{list}\ \alpha'\,(\mathsf{i},\mathsf{j})\ *\ \mathsf{j} \mapsto \mathsf{a},\mathsf{k}) \wedge \alpha = \alpha' \cdot \mathsf{a}\}$

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

$\{\exists \alpha', \mathsf{a}.\ (\mathbf{list}\ \alpha'\,(\mathsf{i},\mathsf{j})\ *\ \mathsf{j} \mapsto \mathsf{a},\mathsf{l}) \wedge \alpha = \alpha' \cdot \mathsf{a}\}$

$\{\mathbf{brlist}\ \alpha\,(\mathsf{i},\mathsf{j},\mathsf{l})\}$

Answer 5.2.3

$\{\mathbf{brlist}\ \alpha\,(\mathsf{i}_1,\mathsf{j}_1,\mathsf{k}_1)\ *\ \mathbf{brlist}\ \beta\,(\mathsf{i}_2,\mathsf{j}_2,\mathsf{k}_2)\}$
$\mathbf{if}\ \mathsf{j}_1 = \mathbf{nil}\ \mathbf{then}\ \mathsf{i}_1 := \mathsf{i}_2\ \mathbf{else}\ [\mathsf{j}_1+1] := \mathsf{i}_2\ ;$
$\{\mathbf{brlist}\ \alpha\,(\mathsf{i}_1,\mathsf{j}_1,\mathsf{i}_2)\ *\ \mathbf{brlist}\ \beta\,(\mathsf{i}_2,\mathsf{j}_2,\mathsf{k}_2)\}$
$\mathbf{if}\ \mathsf{j}_2 = \mathbf{nil}\ \mathbf{then}$
  $\{\mathbf{brlist}\ \alpha\,(\mathsf{i}_1,\mathsf{j}_1,\mathsf{i}_2) \wedge \beta = \epsilon \wedge \mathsf{i}_2 = \mathsf{k}_2\}$
  $\{\mathbf{brlist}\ \alpha{\cdot}\beta\,(\mathsf{i}_1,\mathsf{j}_1,\mathsf{k}_2)\}$
  $\mathsf{j} := \mathsf{j}_1$
$\mathbf{else}$
  $\{\exists\beta', \mathsf{b}.\ (\mathbf{brlist}\ \alpha\,(\mathsf{i}_1,\mathsf{j}_1,\mathsf{i}_2)\ *\ \mathbf{brlist}\ \beta'{\cdot}\mathsf{b}\,(\mathsf{i}_2,\mathsf{j}_2,\mathsf{k}_2)) \wedge \beta = \beta'{\cdot}\mathsf{b}\}$
  $\{\exists\beta', \mathsf{b}.\ (\mathbf{list}\ \alpha\,(\mathsf{i}_1,\mathsf{i}_2)\ *\ \mathbf{list}\ \beta'\,(\mathsf{i}_2,\mathsf{j}_2)\ *\ \mathsf{j}_2 \mapsto \mathsf{b},\mathsf{k}_2) \wedge \beta = \beta'{\cdot}\mathsf{b}\}$
  $\{\exists\beta', \mathsf{b}.\ (\mathbf{list}\ \alpha{\cdot}\beta'\,(\mathsf{i}_1,\mathsf{j}_2)\ *\ \mathsf{j}_2 \mapsto \mathsf{b},\mathsf{k}_2) \wedge \beta = \beta'{\cdot}\mathsf{b}\}$
  $\{\mathbf{brlist}\ \alpha{\cdot}\beta\,(\mathsf{i}_1,\mathsf{j}_2,\mathsf{k}_2)\}$
  $\mathsf{j} := \mathsf{j}_2$
$\{\mathbf{brlist}\ \alpha{\cdot}\beta\,(\mathsf{i}_1,\mathsf{j},\mathsf{k}_2)\}$

Answer 5.2.4

$\{\mathbf{brlist}\ \alpha\,(\mathsf{i},\mathsf{j},\mathsf{k})\}$
if $\mathsf{j} = \mathbf{nil}$ **then**
  $\{\mathbf{emptyheap} \wedge \alpha = \epsilon\}$
  $\mathsf{i} := \mathbf{cons}(\mathsf{a},\mathsf{k})$;
  $\{(\mathbf{emptyheap}\ *\ \mathsf{i} \mapsto \mathsf{a},\mathsf{k}) \wedge \alpha = \epsilon\}$
  $\mathsf{j} := \mathsf{i}$
  $\{(\mathbf{emptyheap}\ *\ \mathsf{j} \mapsto \mathsf{a},\mathsf{k}) \wedge \mathsf{i} = \mathsf{j} \wedge \alpha = \epsilon\}$
  $\{(\mathbf{list}\ \epsilon\,(\mathsf{i},\mathsf{j})\ *\ \mathsf{j} \mapsto \mathsf{a},\mathsf{k}) \wedge \alpha = \epsilon\}$
  $\{\mathbf{brlist}\ \epsilon{\cdot}\mathsf{a}\,(\mathsf{i},\mathsf{j},\mathsf{k}) \wedge \alpha = \epsilon\}$
  $\{\mathbf{brlist}\ \mathsf{a}{\cdot}\epsilon\,(\mathsf{i},\mathsf{j},\mathsf{k}) \wedge \alpha = \epsilon\}$
else
  $\{\exists\beta,\mathsf{b}.\ \mathbf{brlist}\ \beta{\cdot}\mathsf{b}\,(\mathsf{i},\mathsf{j},\mathsf{k}) \wedge \alpha = \beta{\cdot}\mathsf{b}\}$
  $\{\exists\beta,\mathsf{b}.\ (\mathbf{list}\ \beta\,(\mathsf{i},\mathsf{j})\ *\ \mathsf{j} \mapsto \mathsf{b},\mathsf{k}) \wedge \alpha = \beta{\cdot}\mathsf{b}\}$
  $\mathsf{i} := \mathbf{cons}(\mathsf{a},\mathsf{i})$
  $\{\exists\beta,\mathsf{b},\mathsf{i}'.\ (\mathsf{i} \mapsto \mathsf{a},\mathsf{i}'\ *\ \mathbf{list}\ \beta\,(\mathsf{i}',\mathsf{j})\ *\ \mathsf{j} \mapsto \mathsf{b},\mathsf{k}) \wedge \alpha = \beta{\cdot}\mathsf{b}\}$
  $\{\exists\beta,\mathsf{b}.\ (\mathbf{list}\ \mathsf{a}{\cdot}\beta\,(\mathsf{i},\mathsf{j})\ *\ \mathsf{j} \mapsto \mathsf{b},\mathsf{k}) \wedge \alpha = \beta{\cdot}\mathsf{b}\}$
  $\{\exists\beta,\mathsf{b}.\ \mathbf{brlist}\ \mathsf{a}{\cdot}\beta{\cdot}\mathsf{b}\,(\mathsf{i},\mathsf{j},\mathsf{k}) \wedge \alpha = \beta{\cdot}\mathsf{b}\}$
$\{\mathbf{brlist}\ \mathsf{a}{\cdot}\alpha\,(\mathsf{i},\mathsf{j},\mathsf{k})\}$

Answer 5.2.5

$\{\mathbf{brlist}\ \alpha\,(\mathsf{i},\mathsf{j},\mathsf{k})\}$
if $\mathsf{j} = \mathbf{nil}$ **then**
  $\{\mathbf{emptyheap} \wedge \alpha = \epsilon\}$
  $\mathsf{i} := \mathbf{cons}(\mathsf{a},\mathsf{k})$;
  $\{(\mathbf{emptyheap} \wedge \alpha = \epsilon)\ *\ \mathsf{i} \mapsto \mathsf{a},\mathsf{k}\}$
  $\{\mathbf{list}\ \alpha\,(\mathsf{i},\mathsf{i})\ *\ \mathsf{i} \mapsto \mathsf{a},\mathsf{k}\}$
  $\{\mathbf{brlist}\ \alpha{\cdot}\mathsf{a}\,(\mathsf{i},\mathsf{i},\mathsf{k})\}$
  $\mathsf{j} := \mathsf{i}$
else
  $\{\exists\beta,\mathsf{b}.\ \mathbf{brlist}\ \beta{\cdot}\mathsf{b}\,(\mathsf{i},\mathsf{j},\mathsf{k}) \wedge \alpha = \beta{\cdot}\mathsf{b}\}$
  $\{\exists\beta,\mathsf{b}.\ (\mathbf{list}\ \beta\,(\mathsf{i},\mathsf{j})\ *\ \mathsf{j} \mapsto \mathsf{b},\mathsf{k}) \wedge \alpha = \beta{\cdot}\mathsf{b}\}$
  $\mathsf{l} := \mathbf{cons}(\mathsf{a},\mathsf{k})$;
  $\{\exists\beta,\mathsf{b}.\ (\mathbf{list}\ \beta\,(\mathsf{i},\mathsf{j})\ *\ \mathsf{j} \mapsto \mathsf{b},\mathsf{k}\ *\ \mathsf{l} \mapsto \mathsf{a},\mathsf{k}) \wedge \alpha = \beta{\cdot}\mathsf{b}\}$
  $[\mathsf{j}+1] := \mathsf{l}$;
  $\{\exists\beta,\mathsf{b}.\ (\mathbf{list}\ \beta\,(\mathsf{i},\mathsf{j})\ *\ \mathsf{j} \mapsto \mathsf{b},\mathsf{l}\ *\ \mathsf{l} \mapsto \mathsf{a},\mathsf{k}) \wedge \alpha = \beta{\cdot}\mathsf{b}\}$
  $\{\mathbf{list}\ \alpha\,(\mathsf{i},\mathsf{l})\ *\ \mathsf{l} \mapsto \mathsf{a},\mathsf{k}\}$
  $\{\mathbf{brlist}\ \alpha{\cdot}\mathsf{a}\,(\mathsf{i},\mathsf{l},\mathsf{k})\}$
  $\mathsf{j} := \mathsf{l}$
$\{\mathbf{brlist}\ \alpha{\cdot}\mathsf{a}\,(\mathsf{i},\mathsf{j},\mathsf{k})\}$

Answer 5.2.6

$\{\mathbf{brlist}\ \mathsf{a}{\cdot}\alpha\,(\mathsf{i},\mathsf{j},\mathsf{k})\}$
$\{\exists\beta,\mathsf{b}.\ (\mathbf{list}\ \beta\,(\mathsf{i},\mathsf{j}))\ *\ \mathsf{j}\mapsto\mathsf{b},\mathsf{k})\wedge\mathsf{a}{\cdot}\alpha=\beta{\cdot}\mathsf{b}\}$
if $\mathsf{i}=\mathsf{j}$ **then**
  $\{\exists\beta,\mathsf{b}.\ \mathsf{j}\mapsto\mathsf{b},\mathsf{k}\wedge\mathsf{a}{\cdot}\alpha=\beta{\cdot}\mathsf{b}\wedge\beta=\epsilon\}$
  $\{\mathsf{j}\mapsto\mathsf{a},\mathsf{k}\wedge\alpha=\epsilon\}$
  dispose j ; **dispose** $\mathsf{j}+1$;
  $\{\mathbf{emptyheap}\wedge\alpha=\epsilon\}$
  $\mathsf{i}:=\mathsf{k}$; $\mathsf{j}:=\mathbf{nil}$
  $\{\mathbf{emptyheap}\wedge\mathsf{i}=\mathsf{k}\wedge\mathsf{j}=\mathbf{nil}\wedge\alpha=\epsilon\}$
  $\{\mathbf{brlist}\ \epsilon\,(\mathsf{i},\mathsf{j},\mathsf{k})\wedge\alpha=\epsilon\}$
else
  $\{\exists\beta',\mathsf{b},\mathsf{l}.\ (\mathsf{i}\mapsto\mathsf{a},\mathsf{l}\ *\ \mathbf{list}\ \beta'\,(\mathsf{l},\mathsf{j}))\ *\ \mathsf{j}\mapsto\mathsf{b},\mathsf{k})\wedge\mathsf{a}{\cdot}\alpha=\mathsf{a}{\cdot}\beta'{\cdot}\mathsf{b}\}$
  $\{\exists\mathsf{l}.\ \mathsf{i}\mapsto\mathsf{a},\mathsf{l}\ *\ \mathbf{brlist}\ \alpha\,(\mathsf{l},\mathsf{j},\mathsf{k})\}$
  $\mathsf{l}:=[\mathsf{i}+1]$;
  $\{\mathsf{i}\mapsto\mathsf{a},\mathsf{l}\ *\ \mathbf{brlist}\ \alpha\,(\mathsf{l},\mathsf{j},\mathsf{k})\}$
  dispose i ; **dispose** $\mathsf{i}+1$;
  $\{\mathbf{brlist}\ \alpha\,(\mathsf{l},\mathsf{j},\mathsf{k})\}$
  $\mathsf{i}:=\mathsf{l}$
$\{\mathbf{brlist}\ \alpha\,(\mathsf{i},\mathsf{j},\mathsf{k})\}$