An Introduction
to Separation Logic

February 3, 2011

# Chapter 3

# Specifications

From assertions, we move on to specifications, which describe the behavior of commands. In this chapter, we will define the syntax and meaning of specifications, give and illustrate inference rules for proving valid specifications, and define a compact form of proof called an "annotated specification".

Since separation logic has been built upon it, we will review the inference rules of Hoare logic. Further descriptions of this logic, including many examples of proofs, have been given by the author [101, Chapters 1 and 2], [105, Chapters 3 and 4]. A more theoretical view appears in [106, Chapter 8].

The original papers by Hoare [15, 16], as well as earlier work by Naur [18] and Floyd [17], are still well worth reading.

## 3.1 Hoare Triples

For much of these notes, the only kind of specification will be the *Hoare triple*, which consists of two assertions surrounding a command. More precisely, there are two forms of Hoare triple.

A *partial correctness specification*, written

$$\{p\}\ c\ \{q\}$$

is *valid* iff, starting in any state in which the assertion $p$ holds, no execution of the command $c$ aborts and, for any execution of $c$ that terminates in a final state, the assertion $q$ holds in the final state.

A *total correctness specification*, written

$$[\,p\,]\ c\ [\,q\,]$$

is *valid* iff, starting in any state in which $p$ holds, no execution of $c$ aborts, every execution of $c$ terminates, and, for any execution of $c$ that terminates in a final state, $q$ holds in the final state. (In these notes, we will consider total specifications infrequently.)

In both forms, $p$ is called the *precondition* (or *precedent*) and $q$ is called the *postcondition* (or *consequent*).

Notice that, in both forms, there is an implicit universal quantification over both initial and final states. Thus the meaning of a specification is simply **true** or **false**, and does not depend upon a state. Thus to say that a specification is true is the same as saying that it is valid. (This situation will change when we introduce procedures in Section 4.5.)

The following are examples of valid partial correctness specifications of simple commands:

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

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

$$\{\mathbf{emp}\}\ \mathsf{x} := \mathbf{cons}(1, 2)\ \{\mathsf{x} \mapsto 1, 2\}$$

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

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

$$\{\mathsf{x} \mapsto 1, 3 \wedge \mathsf{y} = 1\}\ \mathbf{dispose}\ \mathsf{x}\ \{\mathsf{x} + 1 \mapsto 3 \wedge \mathsf{y} = 1\}$$

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

$$\{\mathbf{true}\}\ \mathbf{while}\ \mathsf{x} \neq 10\ \mathbf{do}\ \mathsf{x} := \mathsf{x} + 1\ \{\mathsf{x} = 10\} \qquad (*)$$

$$\{\mathsf{x} > 10\}\ \mathbf{while}\ \mathsf{x} \neq 10\ \mathbf{do}\ \mathsf{x} := \mathsf{x} + 1\ \{\mathbf{false}\} \qquad (*)$$

A more elaborate example is a specification of the "record-gluing" program (1.5):

```
$\{\mathsf{x} \mapsto - \ * \ \mathsf{y} \mapsto -\}$
$\mathbf{if}\ \mathsf{y} = \mathsf{x} + 1\ \mathbf{then\ skip}$
    $\mathbf{else\ if}\ \mathsf{x} = \mathsf{y} + 1\ \mathbf{then}\ \mathsf{x} := \mathsf{y}$
    $\mathbf{else}\ (\mathbf{dispose}\ \mathsf{x}\ ; \mathbf{dispose}\ \mathsf{y}\ ; \mathsf{x} := \mathbf{cons}(1, 2))$
$\{\mathsf{x} \mapsto -, -\}$
```

(All of the above examples, except those marked $(*)$, would also be valid as total specifications.)

## 3.2 Hoare's Inference Rules for Specifications

As we did with assertions in Section 2.2, we will reason about specifications using inference rules of the form

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

where the premisses and conclusion may contain metavariables, each of which ranges over some set of phrases, such as expressions, variables, or assertions. Again, a rule is said to be *sound* iff, for every instance of the rule, the conclusion of the instance is valid whenever all of the premisses of the instance are valid. But now the premisses may be either specifications or assertions, and the conclusion will be a specification.

In this section, we give the original rules of Hoare logic [15] (along with a more recent rule for variable declarations and several alternative variants of the rules). All of the rules given here remain valid for separation logic. In each case, we give the rule and one or more of its instances:

- Assignment (AS)

$$\frac{}{\{q/v \rightarrow e\}\ v := e\ \{q\}}$$

Instances:

$$\frac{}{\{2 \times \mathsf{y} = 2^{\mathsf{k}+1} \wedge \mathsf{k}+1 \leq \mathsf{n}\}\ \mathsf{k} := \mathsf{k}+1\ \{2 \times \mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\}}$$

$$\frac{}{\{2 \times \mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\}\ \mathsf{y} := 2 \times \mathsf{y}\ \{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\}}$$

- Sequential Composition (SQ)

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

An instance:

$$\frac{\begin{array}{l}\{2 \times \mathsf{y} = 2^{\mathsf{k}+1} \wedge \mathsf{k}+1 \leq \mathsf{n}\}\ \mathsf{k} := \mathsf{k}+1\ \{2 \times \mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\} \\ \qquad\qquad\qquad \{2 \times \mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\}\ \mathsf{y} := 2 \times \mathsf{y}\ \{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\}\end{array}}{\{2 \times \mathsf{y} = 2^{\mathsf{k}+1} \wedge \mathsf{k}+1 \leq \mathsf{n}\}\ \mathsf{k} := \mathsf{k}+1\ ;\ \mathsf{y} := 2 \times \mathsf{y}\ \{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\}}$$

- Strengthening Precedent (SP)

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

An instance:

$$\frac{\begin{array}{l}\mathsf{y} = 2^{\mathsf{k}} \land \mathsf{k} \leq \mathsf{n} \land \mathsf{k} \neq \mathsf{n} \Rightarrow 2 \times \mathsf{y} = 2^{\mathsf{k}+1} \land \mathsf{k} + 1 \leq \mathsf{n} \\ \qquad \{2 \times \mathsf{y} = 2^{\mathsf{k}+1} \land \mathsf{k} + 1 \leq \mathsf{n}\}\ \mathsf{k} := \mathsf{k} + 1\,;\mathsf{y} := 2 \times \mathsf{y}\ \{\mathsf{y} = 2^{\mathsf{k}} \land \mathsf{k} \leq \mathsf{n}\}\end{array}}{\{\mathsf{y} = 2^{\mathsf{k}} \land \mathsf{k} \leq \mathsf{n} \land \mathsf{k} \neq \mathsf{n}\}\ \mathsf{k} := \mathsf{k} + 1\,;\mathsf{y} := 2 \times \mathsf{y}\ \{\mathsf{y} = 2^{\mathsf{k}} \land \mathsf{k} \leq \mathsf{n}\}}$$

In contrast to rules such as Assignment and Sequential Composition, which are called *command-specific rules*, the rules for Strengthening Precedents and Weakening Consequents (to be introduced later) are applicable to arbitrary commands, and are therefore called *structural rules.*

These two rules are exceptional in having premisses, called *verification conditions*, that are assertions rather than specifications. The verification conditions are the mechanism used to introduce mathematical facts about kinds of data, such as $\mathsf{y} = 2^{\mathsf{k}} \land \mathsf{k} \leq \mathsf{n} \land \mathsf{k} \neq \mathsf{n} \Rightarrow 2 \times \mathsf{y} = 2^{\mathsf{k}+1} \land \mathsf{k} + 1 \leq \mathsf{n}$, into proofs of specifications.

To be completely formal, our notion of proof should include formal subproofs of verification conditions, using the rules of predicate calculus as well as rules about integers and other kinds of data. In these notes, however, to avoid becoming mired in proofs of simple arithmetic facts, we will often omit the proofs of verification conditions. It must be emphasized, however, that the soundness of a formal proof can be destroyed by an invalid verification condition.

- Partial Correctness of **while** (WH)

$$\frac{\{i \land b\}\ c\ \{i\}}{\{i\}\ \textbf{while}\ b\ \textbf{do}\ c\ \{i \land \neg\, b\}}$$

An instance:

$$\frac{\{\mathsf{y} = 2^{\mathsf{k}} \land \mathsf{k} \leq \mathsf{n} \land \mathsf{k} \neq \mathsf{n}\}\ \mathsf{k} := \mathsf{k} + 1\,;\mathsf{y} := 2 \times \mathsf{y}\ \{\mathsf{y} = 2^{\mathsf{k}} \land \mathsf{k} \leq \mathsf{n}\}}{\begin{array}{l}\{\mathsf{y} = 2^{\mathsf{k}} \land \mathsf{k} \leq \mathsf{n}\} \\ \textbf{while}\ \mathsf{k} \neq \mathsf{n}\ \textbf{do}\ (\mathsf{k} := \mathsf{k} + 1\,;\mathsf{y} := 2 \times \mathsf{y}) \\ \{\mathsf{y} = 2^{\mathsf{k}} \land \mathsf{k} \leq \mathsf{n} \land \neg\, \mathsf{k} \neq \mathsf{n}\}\end{array}}$$

Here the assertion $i$ is the *invariant* of the **while** command. This is the one inference rule in this chapter that does not extend to total correctness — reflecting that the **while** command is the one construct in our present programming language that can cause nontermination. (In Section 4.5, however, we will find a similar situation when we introduce recursive procedures.)

- Weakening Consequent (WC)

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

An instance:

$$\frac{\begin{array}{l}\{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\}\\ \textbf{while}\ \mathsf{k} \neq \mathsf{n}\ \textbf{do}\ (\mathsf{k} := \mathsf{k} + 1\ ;\ \mathsf{y} := 2 \times \mathsf{y})\\ \{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n} \wedge \neg\, \mathsf{k} \neq \mathsf{n}\}\\[1ex] \mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n} \wedge \neg\, \mathsf{k} \neq \mathsf{n} \Rightarrow \mathsf{y} = 2^{\mathsf{n}}\end{array}}{\begin{array}{l}\{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\}\\ \textbf{while}\ \mathsf{k} \neq \mathsf{n}\ \textbf{do}\ (\mathsf{k} := \mathsf{k} + 1\ ;\ \mathsf{y} := 2 \times \mathsf{y})\\ \{\mathsf{y} = 2^{\mathsf{n}}\}\end{array}}$$

Notice that $\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n} \wedge \neg\, \mathsf{k} \neq \mathsf{n} \Rightarrow \mathsf{y} = 2^{\mathsf{n}}$ is another verification condition.

At this stage, we can give a slightly nontrivial example of a formal proof (of the heart of a simple program for computing powers of two):

1. $\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n} \wedge \mathsf{k} \neq \mathsf{n} \Rightarrow 2 \times \mathsf{y} = 2^{\mathsf{k}+1} \wedge \mathsf{k} + 1 \leq \mathsf{n}$ (VC)

2. $\{2 \times \mathsf{y} = 2^{\mathsf{k}+1} \wedge \mathsf{k} + 1 \leq \mathsf{n}\}\ \mathsf{k} := \mathsf{k} + 1\ \{2 \times \mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\}$ (AS)

3. $\{2 \times \mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\}\ \mathsf{y} := 2 \times \mathsf{y}\ \{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\}$ (AS)

4. $\{2 \times \mathsf{y} = 2^{\mathsf{k}+1} \wedge \mathsf{k} + 1 \leq \mathsf{n}\}\ \mathsf{k} := \mathsf{k} + 1\ ;\ \mathsf{y} := 2 \times \mathsf{y}\ \{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\}$
(SQ 2,3)

5. $\{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n} \wedge \mathsf{k} \neq \mathsf{n}\}\ \mathsf{k} := \mathsf{k} + 1\ ;\ \mathsf{y} := 2 \times \mathsf{y}\ \{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\}$
(SP 1,4)

6. $\{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\}$ (WH 5)
**while** $\mathsf{k} \neq \mathsf{n}$ **do** $(\mathsf{k} := \mathsf{k} + 1\ ;\ \mathsf{y} := 2 \times \mathsf{y})$
$\{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n} \wedge \neg\, \mathsf{k} \neq \mathsf{n}\}$

7. $\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n} \wedge \neg\, \mathsf{k} \neq \mathsf{n} \Rightarrow \mathsf{y} = 2^{\mathsf{n}}$ (VC)

8. $\{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\}$ (WC 6,7)
**while** $\mathsf{k} \neq \mathsf{n}$ **do** $(\mathsf{k} := \mathsf{k} + 1\ ;\ \mathsf{y} := 2 \times \mathsf{y})$
$\{\mathsf{y} = 2^{\mathsf{n}}\}$

Additional rules describe additional forms of commands:

- **skip** (SK)

$$\frac{}{\{q\}\ \textbf{skip}\ \{q\}}$$

An instance:

$$\{\mathsf{y} = 2^{\mathsf{k}} \wedge \neg\, \mathrm{odd}(\mathsf{k})\}\ \textbf{skip}\ \{\mathsf{y} = 2^{\mathsf{k}} \wedge \neg\, \mathrm{odd}(\mathsf{k})\}$$

- Conditional (CD)

$$\frac{\{p \wedge b\}\ c_1\ \{q\} \qquad \{p \wedge \neg\ b\}\ c_2\ \{q\}}{\{p\}\ \textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c_2\ \{q\}}$$

An instance:

$$\frac{\begin{array}{c}\{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathrm{odd}(\mathsf{k})\}\ \mathsf{k} := \mathsf{k}+1\,;\mathsf{y} := 2\times\mathsf{y}\ \{\mathsf{y} = 2^{\mathsf{k}} \wedge \neg\,\mathrm{odd}(\mathsf{k})\}\\ \{\mathsf{y} = 2^{\mathsf{k}} \wedge \neg\,\mathrm{odd}(\mathsf{k})\}\ \mathbf{skip}\ \{\mathsf{y} = 2^{\mathsf{k}} \wedge \neg\,\mathrm{odd}(\mathsf{k})\}\end{array}}{\{\mathsf{y} = 2^{\mathsf{k}}\}\ \mathbf{if}\ \mathrm{odd}(\mathsf{k})\ \mathbf{then}\ \mathsf{k} := \mathsf{k}+1; \mathsf{y} := 2\times\mathsf{y}\ \mathbf{else\ skip}\ \{\mathsf{y} = 2^{\mathsf{k}} \wedge \neg\,\mathrm{odd}(\mathsf{k})\}}$$

- Variable Declaration (DC)

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

when $v$ does not occur free in $p$ or $q$.

An instance:

$$\frac{\begin{array}{l}\{1 = 2^0 \wedge 0 \le \mathsf{n}\}\\ \mathsf{k} := 0\,;\mathsf{y} := 1\,;\\ \mathbf{while}\ \mathsf{k} \neq \mathsf{n}\ \mathbf{do}\ (\mathsf{k} := \mathsf{k}+1\,;\mathsf{y} := 2\times\mathsf{y})\\ \{\mathsf{y} = 2^{\mathsf{n}}\}\end{array}}{\begin{array}{l}\{1 = 2^0 \wedge 0 \le \mathsf{n}\}\\ \mathbf{newvar}\ \mathsf{k}\ \mathbf{in}\\ \quad\big(\mathsf{k} := 0\,;\mathsf{y} := 1\,;\\ \quad\mathbf{while}\ \mathsf{k} \neq \mathsf{n}\ \mathbf{do}\ (\mathsf{k} := \mathsf{k}+1\,;\mathsf{y} := 2\times\mathsf{y})\big)\\ \{\mathsf{y} = 2^{\mathsf{n}}\}\end{array}}$$

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

Notice that the concept of locality is context-dependent: Whether a variable is local or not depends upon the requirements imposed by the surrounding program, which are described by the specification that reflects these requirements. For example, in the specification

$$\{\mathbf{true}\}\ \mathsf{t} := \mathsf{x}+\mathsf{y}\,;\mathsf{y} := \mathsf{t}\times 2\ \{\mathsf{y} = (\mathsf{x}+\mathsf{y})\times 2\},$$

$\mathsf{t}$ is local, and can be declared at the beginning of the command being specified, but in

$$\{\mathbf{true}\}\ \mathsf{t} := \mathsf{x} + \mathsf{y}\ ;\ \mathsf{y} := \mathsf{t} \times 2\ \{\mathsf{y} = (\mathsf{x} + \mathsf{y}) \times 2 \wedge \mathsf{t} = (\mathsf{x} + \mathsf{y})\},$$

$\mathsf{t}$ is not local, and cannot be declared.

For several of the rules we have given, there are alternative versions. For instance:

- Alternative Rule for Assignment (ASalt)

  $$\frac{}{\{p\}\ v := e\ \{\exists v'.\ v = e' \wedge p'\}}$$

  where $v' \notin \{v\} \cup \mathrm{FV}(e) \cup \mathrm{FV}(p)$, $e'$ is $e/v \to v'$, and $p'$ is $p/v \to v'$. The quantifier can be omitted when $v$ does not occur in $e$ or $p$.

The difficulty with this "forward"rule (due to Floyd [17]) is the accumulation of quantifiers, as in the verification condition in the following proof:

1. $\{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} < \mathsf{n}\}\ \mathsf{k} := \mathsf{k} + 1\ \{\exists \mathsf{k}'.\ \mathsf{k} = \mathsf{k}' + 1 \wedge \mathsf{y} = 2^{\mathsf{k}'} \wedge \mathsf{k}' < \mathsf{n}\}$ (ASalt)

2. $\{\exists \mathsf{k}'.\ \mathsf{k} = \mathsf{k}' + 1 \wedge \mathsf{y} = 2^{\mathsf{k}'} \wedge \mathsf{k}' < \mathsf{n}\}\ \mathsf{y} := 2 \times \mathsf{y}$ (ASalt)
   $\{\exists \mathsf{y}'.\ \mathsf{y} = 2 \times \mathsf{y}' \wedge (\exists \mathsf{k}'.\ \mathsf{k} = \mathsf{k}' + 1 \wedge \mathsf{y}' = 2^{\mathsf{k}'} \wedge \mathsf{k}' < \mathsf{n})\}$

3. $\{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} < \mathsf{n}\}\ \mathsf{k} := \mathsf{k} + 1\ ;\ \mathsf{y} := 2 \times \mathsf{y}$ (SQ 1)
   $\{\exists \mathsf{y}'.\ \mathsf{y} = 2 \times \mathsf{y}' \wedge (\exists \mathsf{k}'.\ \mathsf{k} = \mathsf{k}' + 1 \wedge \mathsf{y}' = 2^{\mathsf{k}'} \wedge \mathsf{k}' < \mathsf{n})\}$

4. $(\exists \mathsf{y}'.\ \mathsf{y} = 2 \times \mathsf{y}' \wedge (\exists \mathsf{k}'.\ \mathsf{k} = \mathsf{k}' + 1 \wedge \mathsf{y}' = 2^{\mathsf{k}'} \wedge \mathsf{k}' < \mathsf{n})) \Rightarrow (\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n})$
   (VC)

5. $\{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} < \mathsf{n}\}\ \mathsf{k} := \mathsf{k} + 1\ ;\ \mathsf{y} := 2 \times \mathsf{y}\ \{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\}$ (WC 3,4)

(Compare Step 4 with the verification condition in the previous formal proof.)

- Alternative Rule for Conditionals (CDalt)

  $$\frac{\{p_1\}\ c_1\ \{q\} \qquad \{p_2\}\ c_2\ \{q\}}{\{(b \Rightarrow p_1) \wedge (\neg\ b \Rightarrow p_2)\}\ \mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2\ \{q\}}$$

An instance:

$$\frac{\begin{array}{c}\{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathrm{odd}(\mathsf{k})\}\ \mathsf{k} := \mathsf{k} + 1\ ;\ \mathsf{y} := 2 \times \mathsf{y}\ \{\mathsf{y} = 2^{\mathsf{k}} \wedge \neg\,\mathrm{odd}(\mathsf{k})\} \\ \{\mathsf{y} = 2^{\mathsf{k}} \wedge \neg\,\mathrm{odd}(\mathsf{k})\}\ \textbf{skip}\ \{\mathsf{y} = 2^{\mathsf{k}} \wedge \neg\,\mathrm{odd}(\mathsf{k})\}\end{array}}{\begin{array}{l}\{(\mathrm{odd}(\mathsf{k}) \Rightarrow \mathsf{y} = 2^{\mathsf{k}} \wedge \mathrm{odd}(\mathsf{k})) \wedge (\neg\,\mathrm{odd}(\mathsf{k}) \Rightarrow \mathsf{y} = 2^{\mathsf{k}} \wedge \neg\,\mathrm{odd}(\mathsf{k}))\} \\ \textbf{if}\ \mathrm{odd}(\mathsf{k})\ \textbf{then}\ \mathsf{k} := \mathsf{k} + 1 ; \mathsf{y} := 2 \times \mathsf{y}\ \textbf{else skip} \\ \{\mathsf{y} = 2^{\mathsf{k}} \wedge \neg\,\mathrm{odd}(\mathsf{k})\}\end{array}}$$

(A comparison with the instance of (CD) given earlier indicates why (CD) is usually preferable to (CDalt).)

There is also a rule that combines the rules for strengthening precedents and weakening consequences:

- Consequence (CONSEQ)

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

An instance:

$$\frac{\begin{array}{l}(\mathsf{n} \geq 0) \Rightarrow (1 = 2^0 \wedge 0 \leq \mathsf{n}) \\[1ex] \{1 = 2^0 \wedge 0 \leq \mathsf{n}\} \\ \mathsf{k} := 0\ ;\ \mathsf{y} := 1\ ; \\ \textbf{while}\ \mathsf{k} \neq \mathsf{n}\ \textbf{do}\ (\mathsf{k} := \mathsf{k} + 1\ ;\ \mathsf{y} := 2 \times \mathsf{y}) \\ \{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n} \wedge \neg\, \mathsf{k} \neq \mathsf{n}\} \\[1ex] (\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n} \wedge \neg\, \mathsf{k} \neq \mathsf{n}) \Rightarrow (\mathsf{y} = 2^{\mathsf{n}})\end{array}}{\begin{array}{l}\{\mathsf{n} \geq 0\} \\ \mathsf{k} := 0\ ;\ \mathsf{y} := 1\ ; \\ \textbf{while}\ \mathsf{k} \neq \mathsf{n}\ \textbf{do}\ (\mathsf{k} := \mathsf{k} + 1\ ;\ \mathsf{y} := 2 \times \mathsf{y}) \\ \{\mathsf{y} = 2^{\mathsf{n}}\}\end{array}}$$

(One can use (CONSEQ) in place of (SP) and (WC), but the price is an abundance of vacuous verification conditions of the form $p \Rightarrow p$.)

## 3.3 Annotated Specifications

As was made evident in the previous section, full-blown formal proofs in Hoare logic, even of tiny programs, are long and tedious. Because of this, the usual way of presenting such proofs, at least to human readers, is by means of a specification that has been annotated with intermediate assertions. Several examples of such *annotated specifications* (also often called "proof outlines") have already appeared in Sections 1.5 and 1.11.

The purpose of an annotated specification is to provide enough information so that it would be straightforward for the reader to construct a full formal proof, or at least to determine the verification conditions occuring in the proof. In this section, we will formulate rigorous criteria for achieving this purpose.

In the first place, we note that without annotations, it is not straightforward to construct a proof of a specification from the specification itself. For example, if we try to use the rule for sequential composition,

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

to obtain the main step of a proof of the specification

$$\begin{array}{l} \{\mathsf{n} \geq 0\} \\ (\mathsf{k} := 0\ ;\ \mathsf{y} := 1)\ ; \\ \mathbf{while}\ \mathsf{k} \neq \mathsf{n}\ \mathbf{do}\ (\mathsf{k} := \mathsf{k} + 1\ ;\ \mathsf{y} := 2 \times \mathsf{y}) \\ \{\mathsf{y} = 2^{\mathsf{n}}\}, \end{array}$$

there is no indication of what assertion should replace the metavariable $q$.

But if we were to change the rule to

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

then the new rule would require the annotation $q$ (underlined here for emphasis) to occur in the conclusion:

$$\begin{array}{l}
\{\mathsf{n} \geq 0\} \\
(\mathsf{k} := 0 \,;\, \mathsf{y} := 1) \,; \\
\{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\} \\
\mathbf{while}\ \mathsf{k} \neq \mathsf{n}\ \mathbf{do}\ (\mathsf{k} := \mathsf{k} + 1 \,;\, \mathsf{y} := 2 \times \mathsf{y}) \\
\{\mathsf{y} = 2^{\mathsf{n}}\}.
\end{array}$$

Then, once $q$ has been determined, the premisses must be

$$\begin{array}{l}
\{\mathsf{n} \geq 0\} \\
(\mathsf{k} := 0 \,;\, \mathsf{y} := 1) \,; \\
\{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\}
\end{array}
\qquad \text{and} \qquad
\begin{array}{l}
\{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\} \\
\mathbf{while}\ \mathsf{k} \neq \mathsf{n}\ \mathbf{do} \\
\quad (\mathsf{k} := \mathsf{k} + 1 \,;\, \mathsf{y} := 2 \times \mathsf{y}) \\
\{\mathsf{y} = 2^{\mathsf{n}}\}.
\end{array}$$

The basic trick is to add annotations to the conclusions of the inference rules so that the conclusion of each rule completely determines its premisses. Fortunately, however, it is not necessary to annotate every semicolon in a command — indeed, in many cases one can even omit the precondition (or occasionally the postcondition) from an annotated specification.

The main reason for this state of affairs is that it is often the case that, for a command $c$ and a postcondition $q$, one can calculate a *weakest precondition* $p_w$, which is an assertion such that $\{p\}\ c\ \{q\}$ holds just when $p \Rightarrow p_w$ is valid.

Thus, when we can calculate the weakest precondition $p_w$ of $c$ and $q$, we can regard $c\,\{q\}$ as an annotated specification proving $\{p_w\}\ c\ \{q\}$ — since $c\,\{q\}$ provides enough information to determine $p_w$.

(We are abusing terminology slightly here. For historical reasons, the term weakest precondition is reserved for total correctness, while the phrase *weakest liberal precondition* is used for partial correctness. In these notes, however, since we will rarely consider total correctness, we will drop the qualification "liberal".)

For instance, the weakest precondition of the assignment command $v := e$ and a postcondition $q$ is $q/v \to e$. Thus we can regard $v := e\{q\}$ as an annotated specification for $\{q/v \to e\}\ v := e\ \{q\}$.

In general, we will write the *annotation description*

$$\mathcal{A} \gg \{p\}\ c\ \{q\},$$

and say that $\mathcal{A}$ *establishes* $\{p\}\ c\ \{q\}$, when $\mathcal{A}$ is an annotated specification that determines the specification $\{p\}\ c\ \{q\}$ and shows how to obtain a formal proof of this specification. (The letter $\mathcal{A}$, with various decorations, will be a metavariable ranging over annotated specifications and their subphrases.) We will define the valid annotation descriptions by means of inference rules.

For assignment commands, the inference rule is

- Assignment (ASan)

$$\frac{}{v := e\,\{q\} \gg \{q/v \to e\}\ v := e\ \{q\}.}$$

For example, we have the instances

$$\left.\begin{array}{l} \mathsf{y} := 2 \times \mathsf{y} \\ \{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\} \end{array}\right\} \gg \left\{\begin{array}{l} \{2 \times \mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\} \\ \mathsf{y} := 2 \times \mathsf{y} \\ \{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\} \end{array}\right.$$

and

$$\left.\begin{array}{l} \mathsf{k} := \mathsf{k} + 1 \\ \{2 \times \mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\} \end{array}\right\} \gg \left\{\begin{array}{l} \{2 \times \mathsf{y} = 2^{\mathsf{k}+1} \wedge \mathsf{k} + 1 \leq \mathsf{n}\} \\ \mathsf{k} := \mathsf{k} + 1 \\ \{2 \times \mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\}. \end{array}\right.$$

In general, we say that an annotated specification is *left-complete* if it begins with a precondition, *right-complete* if it ends with a postcondition, and *complete* if it is both left- and right-complete. Then

$$\left.\begin{array}{r} \mathcal{A} \\ \{p\}\mathcal{A} \\ \mathcal{A}\{q\} \\ \{p\}\mathcal{A}\{q\} \end{array}\right\} \text{will match} \left\{\begin{array}{l} \text{any annotated specification.} \\ \text{any left-complete annotated specification.} \\ \text{any right-complete annotated specification.} \\ \text{any complete annotated specification.} \end{array}\right.$$

For the sequential composition of commands, we have the rule:

- Sequential Composition (SQan)

$$\frac{\mathcal{A}_1\,\{q\} \gg \{p\}\ c_1\ \{q\} \qquad \mathcal{A}_2 \gg \{q\}\ c_2\ \{r\}}{\mathcal{A}_1\,;\mathcal{A}_2 \gg \{p\}\ c_1\,;c_2\ \{r\}.}$$

Here the right-complete annotated specification $\mathcal{A}_1\,\{q\}$ in the first premiss must end in the postcondition $\{q\}$, which is stripped from this specification when it occurs within the conclusion. This prevents $\{q\}$ from being duplicated in the conclusion $\mathcal{A}_1 \mathbin{;} \mathcal{A}_2$, or even from occurring there if $\mathcal{A}_2$ is not left-complete.

For example, if we take the above conclusions inferred from (ASan) as premisses, we can infer

$$\left.\begin{array}{l} \mathsf{k} := \mathsf{k}+1 \mathbin{;} \mathsf{y} := 2 \times \mathsf{y} \\ \{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\} \end{array}\right\} \gg \left\{\begin{array}{l} \{2 \times \mathsf{y} = 2^{\mathsf{k}+1} \wedge \mathsf{k}+1 \leq \mathsf{n}\} \\ \mathsf{k} := \mathsf{k}+1 \mathbin{;} \mathsf{y} := 2 \times \mathsf{y} \\ \{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\}. \end{array}\right.$$

Next, there is the rule

- Strengthening Precedent (SPan)

$$\frac{p \Rightarrow q \qquad \mathcal{A} \gg \{q\}\ c\ \{r\}}{\{p\}\,\mathcal{A} \gg \{p\}\ c\ \{r\}.}$$

Notice that this rule can be used to make any assertion left-complete (trivially by using the implication $p \Rightarrow p$). As a nontrivial example, if we take the above conclusion inferred from (SQan) as a premiss, along with the valid verification condition

$$(\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n} \wedge \mathsf{k} \neq \mathsf{n}) \Rightarrow (2 \times \mathsf{y} = 2^{\mathsf{k}+1} \wedge \mathsf{k}+1 \leq \mathsf{n}),$$

we can infer

$$\left.\begin{array}{l} \{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n} \wedge \mathsf{k} \neq \mathsf{n}\} \\ \mathsf{k} := \mathsf{k}+1 \mathbin{;} \mathsf{y} := 2 \times \mathsf{y} \\ \{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\} \end{array}\right\} \gg \left\{\begin{array}{l} \{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n} \wedge \mathsf{k} \neq \mathsf{n}\} \\ \mathsf{k} := \mathsf{k}+1 \mathbin{;} \mathsf{y} := 2 \times \mathsf{y} \\ \{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\}. \end{array}\right.$$

At this point, the reader may wonder whether intermediate assertions are ever actually needed in annotated specifications. In fact, intermediate assertions, and occasionally other annotations, are needed for three reasons:

1. **while** commands and calls of recursive procedures do not always have weakest preconditions that can be expressed in our assertion language.

2. Certain structural inference rules, such as the existential quantification rule (EQ) or the frame rule (FR), do not fit well into the framework of weakest assertions.

3. Intermediate assertions are often needed to simplify verification conditions.

The first of these reasons is illustrated by the rule for the **while** command, whose annotated specifications are required to contain their invariant, immediately before the symbol **while**:

- Partial Correctness of **while** (WHan)

$$\frac{\{i \wedge b\}\ \mathcal{A}\ \{i\} \gg \{i \wedge b\}\ c\ \{i\}}{\{i\}\ \mathbf{while}\ b\ \mathbf{do}\ (\mathcal{A}) \gg \{i\}\ \mathbf{while}\ b\ \mathbf{do}\ c\ \{i \wedge \neg\, b\}.}$$

(Here, the parentheses in the annotated specification $\{i\}$ **while** $b$ **do** $(\mathcal{A})$ are needed to separate postconditions of the body of the **while** command from postconditions of the **while** command itself. A similar usage of parentheses will occur in some other rules for annotation descriptions.)

For example, if we take the above conclusion inferred from (SPan) as a premiss, we can infer

$$\left.\begin{array}{l} \{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\} \\ \mathbf{while}\ \mathsf{k} \neq \mathsf{n}\ \mathbf{do} \\ \quad (\mathsf{k} := \mathsf{k} + 1\, ;\mathsf{y} := 2 \times \mathsf{y}) \end{array}\right\} \gg \left\{\begin{array}{l} \{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\} \\ \mathbf{while}\ \mathsf{k} \neq \mathsf{n}\ \mathbf{do} \\ \quad (\mathsf{k} := \mathsf{k} + 1\, ;\mathsf{y} := 2 \times \mathsf{y}) \\ \{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n} \wedge \neg\, \mathsf{k} \neq \mathsf{n}\}. \end{array}\right.$$

The annotated specifications that can be concluded from the rule (WHan) are not right-complete. However, one can add the postcondition $i \wedge \neg\, b$, or any assertion implied by $i \wedge \neg\, b$, by using the rule

- Weakening Consequent (WCan)

$$\frac{\mathcal{A} \gg \{p\}\ c\ \{q\} \qquad q \Rightarrow r}{\mathcal{A}\,\{r\} \gg \{p\}\ c\ \{r\},}$$

which will make any annotated specification right-complete.

For example, if we take the above conclusion inferred from (WHan) as a premiss, along with the valid verification condition

$$\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n} \wedge \neg\, \mathsf{k} \neq \mathsf{n} \Rightarrow \mathsf{y} = 2^{\mathsf{n}},$$

we obtain

$$\left.\begin{array}{l}\{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\}\\ \textbf{while}\ \mathsf{k} \neq \mathsf{n}\ \textbf{do}\\ \quad(\mathsf{k} := \mathsf{k}+1\,;\mathsf{y} := 2\times\mathsf{y})\\ \{\mathsf{y} = 2^{\mathsf{n}}\}\end{array}\right\} \gg \left\{\begin{array}{l}\{\mathsf{y} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\}\\ \textbf{while}\ \mathsf{k} \neq \mathsf{n}\ \textbf{do}\\ \quad(\mathsf{k} := \mathsf{k}+1\,;\mathsf{y} := 2\times\mathsf{y})\\ \{\mathsf{y} = 2^{\mathsf{n}}\}.\end{array}\right.$$

The reader may verify that further applications of the rules (ASan), (SQan), and (SPan) lead to the annotated specification given at the beginning of this section.

There are additional rules for **skip**, conditional commands, and variable declarations:

- Skip (SKan)

$$\frac{}{\textbf{skip}\,\{q\} \gg \{q\}\ \textbf{skip}\ \{q\}.}$$

- Conditional (CDan)

$$\frac{\{p \wedge b\}\ \mathcal{A}_1\ \{q\} \gg \{p \wedge b\}\ c_1\ \{q\} \qquad \{p \wedge \neg\, b\}\ \mathcal{A}_2\ \{q\} \gg \{p \wedge \neg\, b\}\ c_2\ \{q\}}{\{p\}\ \textbf{if}\ b\ \textbf{then}\ \mathcal{A}_1\ \textbf{else}\ (\mathcal{A}_2)\ \{q\} \gg \{p\}\ \textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c_2\ \{q\}.}$$

- Variable Declaration (DCan)

$$\frac{\{p\}\ \mathcal{A}\ \{q\} \gg \{p\}\ c\ \{q\}}{\{p\}\ \textbf{newvar}\ v\ \textbf{in}\ (\mathcal{A})\ \{q\} \gg \{p\}\ \textbf{newvar}\ v\ \textbf{in}\ c\ \{q\},}$$

  when $v$ does not occur free in $p$ or $q$.

It should be clear that the rules for annotated specifications given in this section allow considerable flexibility in the choice of intermediate assertions. To make proofs clearer, we will often provide more annotations than necessary.

## 3.4 More Structural Inference Rules

In addition to Strengthening Precedent and Weakening Consequent, there are a number of other structural inference rules of Hoare logic that remain sound for separation logic. In this section and the next, we give a variety of these rules. For the moment, we ignore annotated specifications, and give the simpler rules that suffice for formal proofs.

- Vacuity (VAC)

$$\frac{}{\{\textbf{false}\}\ c\ \{q\}}$$

This rule is sound because the definition of both partial and total correctness specifications begin with a universal quantification over states satisfying the precondition — and there are no states satisfying **false**. It is useful for characterizing commands that are not executed (often the bodies of **while** commands). For example, one can prove

1. $(\mathsf{s} = 0 \wedge \mathsf{a} - 1 \geq \mathsf{b} \wedge \mathsf{k} \geq \mathsf{b} \wedge \mathsf{k} < \mathsf{b}) \Rightarrow \textbf{false}$ (VC)

2. $\{\textbf{false}\}$ (VAC)
   $\mathsf{k} := \mathsf{k} + 1\,;\, \mathsf{s} := \mathsf{s} + \mathsf{k}$
   $\{\mathsf{s} = 0 \wedge \mathsf{a} - 1 \geq \mathsf{b} \wedge \mathsf{k} \geq \mathsf{b}\}$

3. $\{\mathsf{s} = 0 \wedge \mathsf{a} - 1 \geq \mathsf{b} \wedge \mathsf{k} \geq \mathsf{b} \wedge \mathsf{k} < \mathsf{b}\}$ (SP)
   $\mathsf{k} := \mathsf{k} + 1\,;\, \mathsf{s} := \mathsf{s} + \mathsf{k}$
   $\{\mathsf{s} = 0 \wedge \mathsf{a} - 1 \geq \mathsf{b} \wedge \mathsf{k} \geq \mathsf{b}\}$

4. $\{\mathsf{s} = 0 \wedge \mathsf{a} - 1 \geq \mathsf{b} \wedge \mathsf{k} \geq \mathsf{b}\}$ (WH)
   **while** $\mathsf{k} < \mathsf{b}$ **do** $(\mathsf{k} := \mathsf{k} + 1\,;\, \mathsf{s} := \mathsf{s} + \mathsf{k})$
   $\{\mathsf{s} = 0 \wedge \mathsf{a} - 1 \geq \mathsf{b} \wedge \mathsf{k} \geq \mathsf{b} \wedge \neg\, \mathsf{k} < \mathsf{b}\}$.

- Disjunction (DISJ)

$$\frac{\{p_1\}\ c\ \{q\} \qquad \{p_2\}\ c\ \{q\}}{\{p_1 \vee p_2\}\ c\ \{q\}}$$

For example, consider two (annotated) specifications of the same command:

$\{\mathsf{a} - 1 \leq \mathsf{b}\}$
$\mathsf{s} := 0\,;\, \mathsf{k} := \mathsf{a} - 1\,;$
$\{\mathsf{s} = \sum_{\mathsf{i}=\mathsf{a}}^{\mathsf{k}} \mathsf{i} \wedge \mathsf{k} \leq \mathsf{b}\}$
**while** $\mathsf{k} < \mathsf{b}$ **do**
  $(\mathsf{k} := \mathsf{k} + 1\,;\, \mathsf{s} := \mathsf{s} + \mathsf{k})$
$\{\mathsf{s} = \sum_{\mathsf{i}=\mathsf{a}}^{\mathsf{b}} \mathsf{i}\}$

$\{\mathsf{a} - 1 \geq \mathsf{b}\}$
$\mathsf{s} := 0\,;\, \mathsf{k} := \mathsf{a} - 1\,;$
$\{\mathsf{s} = 0 \wedge \mathsf{a} - 1 \geq \mathsf{b} \wedge \mathsf{k} \geq \mathsf{b}\}$
**while** $\mathsf{k} < \mathsf{b}$ **do**
  $(\mathsf{k} := \mathsf{k} + 1\,;\, \mathsf{s} := \mathsf{s} + \mathsf{k})$
$\{\mathsf{s} = 0 \wedge \mathsf{a} - 1 \geq \mathsf{b}\}$
$\{\mathsf{s} = \sum_{\mathsf{i}=\mathsf{a}}^{\mathsf{b}} \mathsf{i}\}$.

(Here, the second specification describes the situation where the body of the **while** command is never executed, so that the subspecification of the **while** command can be obtain by using (VAC).) Using these specifications as premisses to (DISJ), we can obtain the main step in

$$\begin{array}{l}\{\textbf{true}\}\\ \{\mathsf{a}-1\leq\mathsf{b}\vee\mathsf{a}-1\geq\mathsf{b}\}\\ \mathsf{s}:=0\,;\mathsf{k}:=\mathsf{a}-1\,;\\ \textbf{while }\mathsf{k}<\mathsf{b}\textbf{ do}\\ \quad(\mathsf{k}:=\mathsf{k}+1\,;\mathsf{s}:=\mathsf{s}+\mathsf{k})\\ \{\mathsf{s}=\sum_{\mathsf{i}=\mathsf{a}}^{\mathsf{b}}\mathsf{i}\}.\end{array}$$

- Conjunction (CONJ)

  $$\frac{\{p_1\}\ c\ \{q_1\}\qquad\{p_2\}\ c\ \{q_2\}}{\{p_1\wedge p_2\}\ c\ \{q_1\wedge q_2\}}$$

  (It should be noted that, in some extensions of separation logic, this rule becomes unsound.)

- Existential Quantification (EQ)

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

  where $v$ is not free in $c$.

- Universal Quantification (UQ)

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

  where $v$ is not free in $c$.

When a variable $v$ does not occur free in the command in a specification (as is required of the premisses of the above two rules), it is said to be a *ghost variable* of the specification.

- Substitution (SUB)

$$\frac{\{p\}\ c\ \{q\}}{\{p/\delta\}\ (c/\delta)\ \{q/\delta\},}$$

where $\delta$ is the substitution $v_1 \rightarrow e_1, \ldots, v_n \rightarrow e_n$, $v_1, \ldots, v_n$ are the variables occurring free in $p$, $c$, or $q$, and, if $v_i$ is modified by $c$, then $e_i$ is a variable that does not occur free in any other $e_j$.

The restrictions on this rule are needed to avoid aliasing. For example, in

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

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

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

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

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

The substitution rule will become important when we consider procedures.

- Renaming (RN)

$$\frac{\{p\}\ \mathbf{newvar}\ v\ \mathbf{in}\ c\ \{q\}}{\{p\}\ \mathbf{newvar}\ v'\ \mathbf{in}\,(c/v \rightarrow v')\ \{q\},}$$

where $v'$ does not occur free in $c$.

Actually, this is not a structural rule, since it is only applicable to variable declarations. On the other hand, unlike the other nonstructural rules, it is not syntax-directed.

The only time it is necessary to use this rule is when one must prove a specification of a variable declaration that violates the proviso that the variable being declared must not occur free in the pre- or postcondition. For example,

1. $\{\mathsf{x} = 0\}\ \mathsf{y} := 1\ \{\mathsf{x} = 0\}$ (AS)
2. $\{\mathsf{x} = 0\}\ \mathbf{newvar}\ \mathsf{y}\ \mathbf{in}\ \mathsf{y} := 1\ \{\mathsf{x} = 0\}$ (DC 1)
3. $\{\mathsf{x} = 0\}\ \mathbf{newvar}\ \mathsf{x}\ \mathbf{in}\ \mathsf{x} := 1\ \{\mathsf{x} = 0\}.$ (RN 2)

In practice, the rule is rarely used. It can usually be avoided by renaming local variables in the program before proving it.

Renaming of bound variables in pre- and postconditions can be treated by using verification conditions such as $(\forall \mathsf{x}.\ \mathsf{z} \neq 2 \times \mathsf{x}) \Rightarrow (\forall \mathsf{y}.\ \mathsf{z} \neq 2 \times \mathsf{y})$ as premisses in the rules (SP) and (WC).

## 3.5 The Frame Rule

In Section 1.5, we saw that the Hoare-logic Rule of Constancy fails for separation logic, but is replaced by the more general

- Frame Rule (FR)

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

  where no variable occurring free in $r$ is modified by $c$.

In fact, the frame rule lies at the heart of separation logic, and provides the key to local reasoning. An instance is

$$\frac{\{\textbf{list}\ \alpha\ \mathsf{i}\}\ \text{“Reverse List”}\ \{\textbf{list}\ \alpha^{\dagger}\ \mathsf{j}\}}{\{\textbf{list}\ \alpha\ \mathsf{i}\ *\ \textbf{list}\ \gamma\ \mathsf{x}\}\ \text{“Reverse List”}\ \{\textbf{list}\ \alpha^{\dagger}\ \mathsf{j}\ *\ \textbf{list}\ \gamma\ \mathsf{x}\},}$$

(assuming “Reverse List” does not modify $\mathsf{x}$ or $\gamma$).

The soundness of the frame rule is surprisingly sensitive to the semantics of our programming language. Suppose, for example, we changed the behavior of deallocation, so that, instead of causing a memory fault, **dispose** $\mathsf{x}$ behaved like **skip** when the value of $\mathsf{x}$ was not in the domain of the heap. Then $\{\textbf{emp}\}$ **dispose** $\mathsf{x}$ $\{\textbf{emp}\}$ would be valid, and the frame rule could be used to infer $\{\textbf{emp}\ *\ \mathsf{x} \mapsto 10\}$ **dispose** $\mathsf{x}$ $\{\textbf{emp}\ *\ \mathsf{x} \mapsto 10\}$. Then, since **emp** is a neutral element for $*$, we would have $\{\mathsf{x} \mapsto 10\}$ **dispose** $\mathsf{x}$ $\{\mathsf{x} \mapsto 10\}$, which is patently false.

To reveal the programming-language properties that the frame rule depends upon, we begin with some definitions:

> If, starting in the state $s, h$, no execution of a command $c$ aborts, then $c$ is *safe at* $s, h$. If, starting in the state $s, h$, every execution of $c$ terminates without aborting, then $c$ *must terminate normally* at $s, h$.

Then the frame rule depends upon two properties of the programming language, which capture the idea that a program will only depend upon or change the part of the initial heap within its footprint, and will abort if any of that part of the heap is missing:

> *Safety Monotonicity* If $\hat{h} \subseteq h$ and $c$ is safe at $s, h - \hat{h}$, then $c$ is safe at $s, h$. If $\hat{h} \subseteq h$ and $c$ must terminate normally at $s, h - \hat{h}$, then $c$ must terminate normally at $s, h$.

*The Frame Property* If $\hat{h} \subseteq h$, $c$ is safe at $s, h - \hat{h}$, and some execution of $c$ starting at $s, h$ terminates normally in the state $s', h'$,

$$\begin{tikzcd}
s, h - \hat{h} \arrow[d, "c \text{ safe}"] & s, h \arrow[l, "\subseteq"'] \arrow[r, "\supseteq"] \arrow[dd, "c"] & \hat{h} \\
{} & & \\
& s', h' &
\end{tikzcd}$$

then $\hat{h} \subseteq h'$ and some execution of $c$ starting at $s, h - \hat{h}$, terminates normally in the state $s', h' - \hat{h}$:

$$\begin{tikzcd}
s, h - \hat{h} \arrow[d, dashed, "c"] & s, h \arrow[l, "\subseteq"'] \arrow[r, "\supseteq"] \arrow[d, "c"] & \hat{h} \\
s', h' - \hat{h} & s', h' \arrow[l, dashed, "\subseteq"'] \arrow[ur, dashed, "\supseteq"'] &
\end{tikzcd}$$

Then:

**Proposition 11** *If the programming language satisfies safety monotonicity and the frame property, then the frame rule is sound for both partial and total correctness.*

Proof Assume $\{p\}\ c\ \{q\}$ is valid, and $s, h \models p \ast r$. Then there is an $\hat{h} \subseteq h$ such that $s, h - \hat{h} \models p$ and $s, \hat{h} \models r$.

From $\{p\}\ c\ \{q\}$, we know that $c$ is safe at $s, h - \hat{h}$ (and in the total-correctness case it must terminate normally). Then, by safety monotonicity, we know that $c$ is safe at $s, h$ (and in the total-correctness case it must terminate normally).

Suppose that, starting in the state $s, h$, there is an execution of $c$ that terminates in the state $s', h'$. Since $c$ is safe at $s, h - \hat{h}$, we know by the frame property that $\hat{h} \subseteq h'$ and that, starting in the state $s, h - \hat{h}$, there is some execution of $c$ that terminates in the state $s', h' - \hat{h}$. Then $\{p\}\ c\ \{q\}$ and $s, h - \hat{h} \models p$ imply that $s', h' - \hat{h} \models q$.

Since an execution of $c$ carries $s, h$ into $s', h'$, we know that $s\,v = s'v$ for all $v$ that are not modified by $c$. Then, since these include the free variables of $r$, $s, \hat{h} \models r$ implies that $s', \hat{h} \models r$. Thus $s', h' \models q \ast r$. end of proof

## 3.6 More Rules for Annotated Specifications

We now consider how to enlarge the concept of annotated specifications to encompass the structural rules given in the two preceding sections. Since these rules are applicable to arbitrary commands, their annotated versions will indicate explicitly which rule is being applied.

In all of these rules, we assume that the annotated specifications in the premisses will often be sequences of several lines. In the unary rules (VACan), (EQan), (UQan), (FRan), and (SUBan), braces are used to indicate the vertical extent of the single operand. In the binary rules (DISJan), and (CONJan), the two operands are placed symmetrically around the indicator DISJ or CONJ.

For the vacuity rule, we have

- Vacuity (VACan)

$$\frac{}{\Big\{c\Big\}\text{VAC}\,\{q\} \gg \{\mathbf{false}\}\ c\ \{q\}.}$$

Here $c$ contains no annotations, since no reasoning about its subcommands is used. For example, using (VACan), (SPan), (WHan), and (WCan):

$$\begin{array}{l}
\{\mathsf{s} = 0 \land \mathsf{a} - 1 \geq \mathsf{b} \land \mathsf{k} \geq \mathsf{b}\} \\
\mathbf{while}\ \mathsf{k} < \mathsf{b}\ \mathbf{do} \\
\quad \left.\begin{array}{l} (\mathsf{k} := \mathsf{k} + 1\,; \\ \mathsf{s} := \mathsf{s} + \mathsf{k}) \end{array}\right\}\text{VAC} \\
\{\mathsf{s} = 0 \land \mathsf{a} - 1 \geq \mathsf{b}\}.
\end{array}$$

Notice that, here and with later unary structural rules, when the braced sequence contains several lines, we will omit the left brace. For the disjunction rule, we have

- Disjunction (DISJan)

$$\frac{\mathcal{A}_1\,\{q\} \gg \{p_1\}\ c\ \{q\} \qquad \mathcal{A}_2\,\{q\} \gg \{p_2\}\ c\ \{q\}}{(\mathcal{A}_1\ \text{DISJ}\ \mathcal{A}_2)\,\{q\} \gg \{p_1 \lor p_2\}\ c\ \{q\}.}$$

For example,

$$\begin{array}{lcl}
 & \{\mathbf{true}\} & \\
\begin{array}{l} \{\mathsf{a}-1 \le \mathsf{b}\} \\ \mathsf{s} := 0\,;\ \mathsf{k} := \mathsf{a}-1\,; \\ \{\mathsf{s} = \sum_{\mathsf{i}=\mathsf{a}}^{\mathsf{k}} \mathsf{i} \wedge \mathsf{k} \le \mathsf{b}\} \\ \mathbf{while}\ \mathsf{k} < \mathsf{b}\ \mathbf{do} \\ \quad (\mathsf{k} := \mathsf{k}+1\,;\ \mathsf{s} := \mathsf{s}+\mathsf{k}) \end{array} & \mathrm{DISJ} & \begin{array}{l} \{\mathsf{a}-1 \ge \mathsf{b}\} \\ \mathsf{s} := 0\,;\ \mathsf{k} := \mathsf{a}-1\,; \\ \{\mathsf{s} = 0 \wedge \mathsf{a}-1 \ge \mathsf{b} \wedge \mathsf{k} \ge \mathsf{b}\} \\ \mathbf{while}\ \mathsf{k} < \mathsf{b}\ \mathbf{do} \\ \quad (\mathsf{k} := \mathsf{k}+1\,;\ \mathsf{s} := \mathsf{s}+\mathsf{k})\Big\}\mathrm{VAC} \\ \{\mathsf{s} = 0 \wedge \mathsf{a}-1 \ge \mathsf{b}\}. \end{array} \\
 & \{\mathsf{s} = \sum_{\mathsf{i}=\mathsf{a}}^{\mathsf{b}} \mathsf{i}\}. &
\end{array}$$

In the remaining structural rules, the annotated specification in the conclusion need not be left- or right-complete; it simply contains the annotated specifications of the premisses.

- Conjunction (CONJan)

$$\frac{\mathcal{A}_1 \gg \{p_1\}\ c\ \{q_1\} \qquad \mathcal{A}_2 \gg \{p_2\}\ c\ \{q_2\}}{(\mathcal{A}_1\ \mathrm{CONJ}\ \mathcal{A}_2) \gg \{p_1 \wedge p_2\}\ c\ \{q_1 \wedge q_2\}.}$$

- Existential Quantification (EQan)

$$\frac{\mathcal{A} \gg \{p\}\ c\ \{q\}}{\Big\{\mathcal{A}\Big\}\,\exists v\ \gg \{\exists v.\ p\}\ c\ \{\exists v.\ q\},}$$

  where $v$ is not free in $c$.

- Universal Quantification (UQan)

$$\frac{\mathcal{A} \gg \{p\}\ c\ \{q\}}{\Big\{\mathcal{A}\Big\}\,\forall v\ \gg \{\forall v.\ p\}\ c\ \{\forall v.\ q\},}$$

  where $v$ is not free in $c$.

(In using the two rules above, we will often abbreviate $\Big\{\cdots\Big\{\mathcal{A}\Big\}\exists v_1 \cdots\Big\}\exists v_n$ by $\Big\{\mathcal{A}\Big\}\exists v_1.\ldots, v_n$, and $\Big\{\cdots\Big\{\mathcal{A}\Big\}\forall v_1 \cdots\Big\}\forall v_n$ by $\Big\{\mathcal{A}\Big\}\forall v_1.\ldots, v_n$.)

- Frame (FRan)

$$\frac{\mathcal{A} \gg \{p\}\ c\ \{q\}}{\Big\{\mathcal{A}\Big\} \ast r \gg \{p \ast r\}\ c\ \{q \ast r\},}$$

  where no variable occurring free in $r$ is modified by $c$.

For example,

$$\begin{array}{l} \{\exists \mathsf{j}.\ \mathsf{x} \mapsto -, \mathsf{j} \ast \mathsf{list}\,\alpha\,\mathsf{j}\} \\ \left.\begin{array}{l} \left.\begin{array}{l} \{\mathsf{x} \mapsto -\} \\ [\mathsf{x}] := \mathsf{a} \\ \{\mathsf{x} \mapsto \mathsf{a}\} \end{array}\right\} \ast \mathsf{x}+1 \mapsto \mathsf{j} \ast \mathsf{list}\,\alpha\,\mathsf{j} \end{array}\right\} \exists \mathsf{j} \\ \{\exists \mathsf{j}.\ \mathsf{x} \mapsto \mathsf{a}, \mathsf{j} \ast \mathsf{list}\,\alpha\,\mathsf{j}\} \end{array}$$

- Substitution (SUBan)

$$\frac{\mathcal{A} \gg \{p\}\ c\ \{q\}}{\Big\{\mathcal{A}\Big\}/\delta \gg \{p/\delta\}\ (c/\delta)\ \{q/\delta\},}$$

  where $\delta$ is the substitution $v_1 \to e_1, \ldots, v_n \to e_n$, $v_1, \ldots, v_n$ are the variables occurring free in $p$, $c$, or $q$, and, if $v_i$ is modified by $c$, then $e_i$ is a variable that does not occur free in any other $e_j$.

In the conclusion of this rule, $\Big\{\mathcal{A}\Big\}/\delta$ denotes an annotated specification in which "/" and the substitution denoted by $\delta$ occur literally, i.e., the substitution is not carried out on $\mathcal{A}$. The total substitution $\delta$ may be abbreviated by a partial substitution, but the conditions on what is substituted for the $v_i$ must hold for all $v_1, \ldots, v_n$ occurring free in $p$, $c$, or $q$.

We omit any discussion of the renaming rule (RN), since it is rarely used, and does not lend itself to annotated specifications.

## 3.7 Inference Rules for Mutation and Disposal

Finally, we come to the new commands for manipulating the heap, which give rise to a surprising variety of inference rules. For each of these four commands, we can give three kinds of inference rule: local, global, and backward-reasoning.

In this and the next two section, we will present these rules and show that, for each type of command, the different rules are all derivable from one another (except for specialized rules that are only applicable in the "nonoverwriting" case). (Some of the deriviations will make use of the inference rules for assertions derived in Section 2.4.)

For mutation commands, we have

- The local form (MUL)

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

- The global form (MUG)

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

- The backward-reasoning form (MUBR)

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

One can derive (MUG) from (MUL) by using the frame rule:

$$\begin{array}{l}
\{(e \mapsto -) \ast r\} \\
\quad \left.\begin{array}{l}
\{e \mapsto -\} \\
[e] := e' \\
\{e \mapsto e'\}
\end{array}\right\} \ast r \\
\{(e \mapsto e') \ast r\},
\end{array}$$

while to go in the opposite direction it is only necessary to take $r$ to be **emp**:

$$\begin{array}{l}
\{e \mapsto -\} \\
\{(e \mapsto -) \ast \mathbf{emp}\} \\
[e] := e' \\
\{(e \mapsto e') \ast \mathbf{emp}\} \\
\{e \mapsto e'\}.
\end{array}$$

To derive (MUBR) from (MUG), we take $r$ to be $(e \mapsto e') -\!\!* \; p$ and use the derived axiom schema (2.5): $q * (q -\!\!* \; p) \Rightarrow p$.

$$\begin{array}{l} \{(e \mapsto -) * ((e \mapsto e') -\!\!* \; p)\} \\ [e] := e' \\ \{(e \mapsto e') * ((e \mapsto e') -\!\!* \; p)\} \\ \{p\}, \end{array}$$

while to go in the opposite direction we take $p$ to be $(e \mapsto e') * r$ and use the derived axiom schema (2.7): $(p * r) \Rightarrow (p * (q -\!\!* \; (q * r)))$.

$$\begin{array}{l} \{(e \mapsto -) * r\} \\ \{(e \mapsto -) * ((e \mapsto e') -\!\!* \; ((e \mapsto e') * r))\} \\ [e] := e' \\ \{(e \mapsto e') * r\}. \end{array}$$

For deallocation, there are only two rules, since the global form is also suitable for backward reasoning:

- The local form (DISL)

$$\frac{}{\{e \mapsto -\} \; \mathbf{dispose} \; e \; \{\mathbf{emp}\}.}$$

- The global (and backward-reasoning) form (DISBR)

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

As with the mutation rules, one can derive (DISBR) from (DISL) by using the frame rule, and go in the opposite direction by taking $r$ to be **emp**.

## 3.8 Rules for Allocation

When we turn to the inference rules for allocation and lookup, our story becomes more complicated, since these commands modify variables. More precisely, they are what we will call *generalized assignment commands*, i.e., commands that first perform a computation that does not alter the store (though it may affect the heap), and then after this computation has finished, change the value of the store for a single variable.

However, neither allocation nor lookup are assignment commands in the sense we will use in these notes, since they do not obey the rule (AS) for assignment. For example, if we tried to apply this rule to an allocation command, we could obtain

$$\{\mathbf{cons}(1,2) = \mathbf{cons}(1,2)\}\ \mathsf{x} := \mathbf{cons}(1,2)\ \{\mathsf{x} = \mathsf{x}\}, \qquad \text{(syntactically illegal)}$$

where the precondition is not a syntactically well-formed assertion, since $\mathbf{cons}(1,2)$ is not an expression — for the compelling reason that it has a side effect.

Even in the analogous case for lookup,

$$\{[\mathsf{y}] = [\mathsf{y}]\}\ \mathsf{x} := [\mathsf{y}]\ \{\mathsf{x} = \mathsf{x}\} \qquad \text{(syntactically illegal)}$$

is prohibited since $[\mathsf{y}]$ can have the side effect of aborting.

For allocation (and lookup) it is simplest to begin with local and global rules for the *nonoverwriting* case, where the old value of the variable being modified plays no role. For brevity, we abbreviate the sequence $e_1, \ldots, e_n$ of expressions by $\overline{e}$:

- The local nonoverwriting form (CONSNOL)

  $$\frac{}{\{\mathbf{emp}\}\ v := \mathbf{cons}(\overline{e})\ \{v \mapsto \overline{e}\}},$$

  where $v \notin \mathrm{FV}(\overline{e})$.

- The global nonoverwriting form (CONSNOG)

  $$\frac{}{\{r\}\ v := \mathbf{cons}(\overline{e})\ \{(v \mapsto \overline{e})\ *\ r\}},$$

  where $v \notin \mathrm{FV}(\overline{e}, r)$.

As with mutation and deallocation, one can derive the global form from the local by using the frame rule, and the local from the global by taking $r$ to be **emp**.

The price for the simplicity of the above rules is the prohibition of overwriting, which is expressed by the conditions $v \notin \mathrm{FV}(\overline{e})$ and $v \notin \mathrm{FV}(\overline{e}, r)$. Turning to the more complex and general rules that permit overwriting, we have three forms for allocation:

- The local form (CONSL)

$$\frac{}{\{v = v' \wedge \mathbf{emp}\}\ v := \mathbf{cons}(\overline{e})\ \{v \mapsto \overline{e}'\}},$$

  where $v'$ is distinct from $v$, and $\overline{e}'$ denotes $\overline{e}/v \rightarrow v'$ (i.e., each $e_i'$ denotes $e_i/v \rightarrow v'$).

- The global form (CONSG)

$$\frac{}{\{r\}\ v := \mathbf{cons}(\overline{e})\ \{\exists v'.\ (v \mapsto \overline{e}') \ast r'\}},$$

  where $v'$ is distinct from $v$, $v' \notin \mathrm{FV}(\overline{e}, r)$, $\overline{e}'$ denotes $\overline{e}/v \rightarrow v'$, and $r'$ denotes $r/v \rightarrow v'$.

- The backward-reasoning form (CONSBR)

$$\frac{}{\{\forall v''.\ (v'' \mapsto \overline{e}) \mathrel{-\!\!\ast} p''\}\ v := \mathbf{cons}(\overline{e})\ \{p\}},$$

  where $v''$ is distinct from $v$, $v'' \notin \mathrm{FV}(\overline{e}, p)$, and $p''$ denotes $p/v \rightarrow v''$.

To explain these rules, we begin with (CONSG). Here the existentially quantified variable $v'$ denotes the old value of $v$, which has been overwritten by the allocation command (and may possibly no longer be determined by the store), much as in the alternative assignment rule (ASalt) in Section 3.2. A typical instance is

$$\{\mathsf{list}\ \alpha\ \mathsf{i}\}\ \mathsf{i} := \mathbf{cons}(3, \mathsf{i})\ \{\exists \mathsf{j}.\ \mathsf{i} \mapsto 3, \mathsf{j} \ast \mathsf{list}\ \alpha\ \mathsf{j}\}.$$

One can derive (CONSG) from the nonoverwriting rule (CONSNOG) by using a plausible equivalence that captures the essence of generalized assignment:

$$v := \mathbf{cons}(\overline{e}) \cong \mathbf{newvar}\ \hat{v}\ \mathbf{in}\ (\hat{v} := \mathbf{cons}(\overline{e})\ ; v := \hat{v}), \tag{3.1}$$

where $\hat{v}$ does not occur in $\overline{e}$ — and can be chosen not to occur in other specified phrases. (Here $\cong$ denotes an equivalence of meaning between two commands.)

We can regard this equivalence as defining the possibly overwriting case of allocation on the left by the nonoverwriting case on the right. Then we can derive (CONSG) from (CONSNOG) by existentially quantifying $v$ and renaming it to $v'$ (using the last axiom schema displayed in 2.2 in Section 2.2: $(p/v \to e) \Rightarrow (\exists v.\ p)$).

$$\begin{array}{l}
\{r\} \\
\mathbf{newvar}\ \hat{v}\ \mathbf{in} \\
\quad \big(\hat{v} := \mathbf{cons}(\overline{e})\ ; \\
\quad \{(\hat{v} \mapsto \overline{e})\ *\ r\} \\
\quad \{\exists v.\ (\hat{v} \mapsto \overline{e})\ *\ r\} \\
\quad \{\exists v'.\ (\hat{v} \mapsto \overline{e}')\ *\ r'\} \\
\quad v := \hat{v}\big) \\
\{\exists v'.\ (v \mapsto \overline{e}')\ *\ r'\}
\end{array}$$

One might expect the local rule to be

$$\frac{}{\{\mathbf{emp}\}\ v := \mathbf{cons}(\overline{e})\ \{\exists v'.\ (v \mapsto \overline{e}')\},}$$

(where $v'$ is distinct from $v$ and $v' \notin \mathrm{FV}(\overline{e})$), which can be derived from (CONSG) by taking $r$ to be **emp**. But this rule, though sound, is too weak. For example, the postcondition of the instance

$$\{\mathbf{emp}\}\ \mathsf{i} := \mathbf{cons}(3, \mathsf{i})\ \{\exists \mathsf{j}.\ \mathsf{i} \mapsto 3, \mathsf{j}\}$$

gives no information about the second component of the new record.

In the stronger local rule (CONSL), the existential quantifier is dropped and $v'$ becomes a variable that is not modified by $v := \mathbf{cons}(\overline{e})$, so that its occurrences in the postcondition denote the same value as in the precondition.

For example, the following instance of (CONSL)

$$\{\mathsf{i} = \mathsf{j} \wedge \mathbf{emp}\}\ \mathsf{i} := \mathbf{cons}(3, \mathsf{i})\ \{\mathsf{i} \mapsto 3, \mathsf{j}\}$$

shows that the value of $\mathsf{j}$ in the postcondition is the value of $\mathsf{i}$ before the assignment.

We can derive (CONSL) from (CONSG) by replacing $r$ by $v = v' \wedge \mathbf{emp}$ and $v'$ by $v''$, and using the fact that $v'' = v'$ is pure, plus simple properties of equality and the existential quantifier:

$$\begin{array}{l}
\{v = v' \wedge \mathbf{emp}\}\\
v := \mathbf{cons}(\overline{e})\\
\{\exists v''.\ (v \mapsto \overline{e}'') \ast (v'' = v' \wedge \mathbf{emp})\}\\
\{\exists v''.\ ((v \mapsto \overline{e}'') \wedge v'' = v') \ast \mathbf{emp}\}\\
\{\exists v''.\ (v \mapsto \overline{e}'') \wedge v'' = v'\}\\
\{\exists v''.\ (v \mapsto \overline{e}') \wedge v'' = v'\}\\
\{v \mapsto \overline{e}'\}.
\end{array}$$

(Here $v''$ is chosen to be distinct from $v$, $v'$, and the free variables of $\overline{e}$.)

Then to complete the circle (and show the adequacy of (CONSL)), we derive (CONSG) from (CONSL) by using the frame rule and (EQ):

$$\begin{array}{l}
\{r\}\\
\{\exists v'.\ v = v' \wedge r\}\\
\quad \left.\begin{array}{l}
\{v = v' \wedge r\}\\
\{v = v' \wedge r'\}\\
\{v = v' \wedge (\mathbf{emp} \ast r')\}\\
\{(v = v' \wedge \mathbf{emp}) \ast r'\}\\
\quad \left.\begin{array}{l}
\{(v = v' \wedge \mathbf{emp})\}\\
v := \mathbf{cons}(\overline{e})\\
\{(v \mapsto \overline{e}')\}
\end{array}\right\} \ast\ r'\\
\{(v \mapsto \overline{e}') \ast r'\}
\end{array}\right\} \exists v'\\
\{\exists v'.\ (v \mapsto \overline{e}') \ast r'\}
\end{array}$$

In the backward-reasoning rule (CONSBR), the universal quantifier $\forall v''$ in the precondition expresses the nondeterminacy of allocation. (We have chosen the metavariable $v''$ rather than $v'$ to simplify some derivations.) The most direct way to see this is by a semantic proof that the rule is sound:

Suppose the precondition holds in the state $s, h$, i.e., that

$$s, h \models \forall v''.\ (v'' \mapsto \overline{e}) \mathrel{-\!\!\ast} p''.$$

Then the semantics of universal quantification gives

$$\forall \ell.\ [\, s \mid v''\!: \ell \,], h \models (v'' \mapsto \overline{e}) \mathrel{-\!\!\ast} p'',$$

and the semantics of separating implication gives

$$\forall \ell, h'.\ h \perp h' \text{ and } \underline{[\, s \mid v'' \colon \ell\,], h' \models (v'' \mapsto \overline{e})} \text{ implies } [\, s \mid v'' \colon \ell\,], h \cdot h' \models p'',$$

where the underlined formula is equivalent to

$$h' = [\, \ell \colon [\![e_1]\!]_{\mathrm{exp}} s \mid \ldots \mid \ell + n - 1 \colon [\![e_n]\!]_{\mathrm{exp}} s\,].$$

Thus

$$\begin{aligned}&\forall \ell.\ \big(\ell, \ldots, \ell + n - 1 \notin \operatorname{dom} h \text{ implies}\\ &\qquad [\, s \mid v'' \colon \ell\,], [\, h \mid \ell \colon [\![e_1]\!]_{\mathrm{exp}} s \mid \ldots \mid \ell + n - 1 \colon [\![e_n]\!]_{\mathrm{exp}} s\,] \models p''\big).\end{aligned}$$

Then, by Proposition 3 in Chapter 2, since $p''$ denotes $p/v \to v''$, we have $[\, s \mid v'' \colon \ell\,], h{\cdot}h' \models p''$ iff $\hat{s}, h{\cdot}h' \models p$, where

$$\hat{s} = [\, s \mid v'' \colon \ell \mid v \colon [\![v'']\!]_{\mathrm{exp}} [\, s \mid v'' \colon \ell\,]\,] = [\, s \mid v'' \colon \ell \mid v \colon \ell\,].$$

Moreover, since $v''$ does not occur free in $p$, we can simplify $\hat{s}, h{\cdot}h' \models p$ to $[\, s \mid v \colon \ell\,], h{\cdot}h' \models p$. Thus

$$\begin{aligned}&\forall \ell.\ \big(\ell, \ldots, \ell + n - 1 \notin \operatorname{dom} h \text{ implies}\\ &\qquad [\, s \mid v \colon \ell\,], [\, h \mid \ell \colon [\![e_1]\!]_{\mathrm{exp}} s \mid \ldots \mid \ell + n - 1 \colon [\![e_n]\!]_{\mathrm{exp}} s\,] \models p\big).\end{aligned}$$

Now execution of the allocation command $v := \mathbf{cons}(\overline{e})$, starting in the state $s, h$, will never abort, and will always termininate in a state $[\, s \mid v \colon \ell\,], [\, h \mid \ell \colon [\![e_1]\!]_{\mathrm{exp}} s \mid \ldots \mid \ell + n - 1 \colon [\![e_n]\!]_{\mathrm{exp}} s\,]$ for some $\ell$ such that $\ell, \ldots, \ell + n - 1 \notin \operatorname{dom} h$. Thus the condition displayed above insures that all possible terminating states satisfy the postcondition $p$.

We also show that (CONSBR) and (CONSG) are interderivable. To derive (CONSBR) from (CONSG), we choose $v' \notin \mathrm{FV}(\overline{e}, p)$ to be distinct from $v$ and $v''$, take $r$ to be $\forall v''.\ (v'' \mapsto \overline{e}) \mathrel{-\!\!*} p''$, and use predicate-calculus properties of quantifiers, as well as (2.5): $q \ast (q \mathrel{-\!\!*} p) \Rightarrow p$.

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

To go in the other direction, we choose $v'' \notin \mathrm{FV}(\overline{e}, r)$ to be distinct from $v$ and $v'$, take $p$ to be $\exists v'.\ (v \mapsto \overline{e}') \ast r'$, and use properties of quantifiers, as well as (2.6): $r \Rightarrow (q \mathrel{-\!\!\ast} (q \ast r))$.

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

## 3.9 Rules for Lookup

Finally, we come to the lookup command, which — for no obvious reason — has the richest variety of inference rules. We begin with the nonoverwriting rules:

- The local nonoverwriting form (LKNOL)

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

  where $v \notin \mathrm{FV}(e)$.

- The global nonoverwriting form (LKNOG)

$$\frac{}{\{\exists v''.\ (e \mapsto v'') \ast p''\}\ v := [e]\ \{(e \mapsto v) \ast p\},}$$

  where $v \notin \mathrm{FV}(e)$, $v'' \notin \mathrm{FV}(e) \cup (\mathrm{FV}(p) - \{v\})$, and $p''$ denotes $p/v \to v''$.

In (LKNOG), there is no restriction preventing $v''$ from being the same variable as $v$. Thus, as a special case,

$$\frac{}{\{\exists v.\ (e \mapsto v) \ast p\}\ v := [e]\ \{(e \mapsto v) \ast p\},}$$

where $v \notin \mathrm{FV}(e)$. For example, if we take

$$\begin{array}{lll}
v & \text{to be} & \mathsf{j} \\
e & \text{to be} & \mathsf{i} + 1
\end{array}
\qquad
p \quad \text{to be} \quad \mathsf{i} \mapsto 3 \ast \mathbf{list}\ \alpha\ \mathsf{j},$$

(and remember that $\mathsf{i} \mapsto 3, \mathsf{j}$ abbreviates $(\mathsf{i} \mapsto 3) \ast (\mathsf{i} + 1 \mapsto \mathsf{j})$), then we obtain the instance

$$\begin{array}{l} \{\exists \mathsf{j}.\ \mathsf{i} \mapsto 3, \mathsf{j} \ast \mathbf{list}\ \alpha\ \mathsf{j}\} \\ \mathsf{j} := [\mathsf{i} + 1] \\ \{\mathsf{i} \mapsto 3, \mathsf{j} \ast \mathbf{list}\ \alpha\ \mathsf{j}\}. \end{array}$$

In effect, the action of the lookup command is to erase an existential quantifier. In practice, if one chooses the names of quantified variables with foresight, most lookup commands can be described by this simple special case.

Turning to the rules for the general case of lookup, we have

- The local form (LKL)

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

  where $v$, $v'$, and $v''$ are distinct, and $e'$ denotes $e/v \to v'$.

- The global form (LKG)

  $$\frac{}{\begin{array}{l} \{\exists v''.\ (e \mapsto v'') \ast (r/v' \to v)\}\ v := [e] \\ \qquad\qquad\qquad \{\exists v'.\ (e' \mapsto v) \ast (r/v'' \to v)\}, \end{array}}$$

  where $v$, $v'$, and $v''$ are distinct, $v'$, $v'' \notin \mathrm{FV}(e)$, $v \notin \mathrm{FV}(r)$, and $e'$ denotes $e/v \to v'$.

- The first backward-reasoning form (LKBR1)

  $$\frac{}{\{\exists v''.\ (e \mapsto v'') \ast ((e \mapsto v'') -\!\!\ast\ p'')\}\ v := [e]\ \{p\},}$$

  where $v'' \notin \mathrm{FV}(e) \cup (\mathrm{FV}(p) - \{v\})$, and $p''$ denotes $p/v \to v''$.

- The second backward-reasoning form (LKBR2)

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

  where $v'' \notin \mathrm{FV}(e) \cup (\mathrm{FV}(p) - \{v\})$, and $p''$ denotes $p/v \to v''$.

In each of these rules, one can think of $v'$ as denoting the value of $v$ before execution of the lookup command, and $v''$ as denoting the value of $v$ after execution.

We begin with a semantic proof of the soundness of the local rule (LKL). Suppose that the precondition holds in the state $s_0, h$, i.e., that

$$s_0, h \models v = v' \wedge (e \mapsto v'').$$

Then $s_0\, v = s_0\, v'$ and $h = [\, [\![e]\!]_{\mathrm{exp}} s_0 \colon s_0\, v'' \,]$.

Starting in the state $s_0, h$, the execution of $v := [e]$ will not abort (since $[\![e]\!]_{\mathrm{exp}} s_0 \in \mathrm{dom}\, h$), and will terminate with the store $s_1 = [\, s_0 \mid v \colon s_0\, v'' \,]$ and the unchanged heap $h$. To see that this state satisfies the postcondition, we note that $s_1\, v = s_0\, v'' = s_1\, v''$ and, since $e'$ does not contain $v$, $[\![e']\!]_{\mathrm{exp}} s_1 = [\![e']\!]_{\mathrm{exp}} s_0$. Then by applying Proposition 3 in Chapter 2, with $\hat{s} = [\, s_0 \mid v \colon s_0\, v' \,] = [\, s_0 \mid v \colon s_0\, v \,] = s_0$, we obtain $[\![e']\!]_{\mathrm{exp}} s_0 = [\![e]\!]_{\mathrm{exp}} s_0$. Thus $h = [\, [\![e']\!]_{\mathrm{exp}} s_1 \colon s_1\, v \,]$, and $s_1, h \models v = v'' \wedge (e' \mapsto v)$.

To derive (LKG) from (LKL), we use the frame rule and two applications of (EQ):

$$
\begin{array}{l}
\{\exists v''.\ (e \mapsto v'') \ * \ (r/v' \to v)\}\\
\{\exists v', v''.\ (v = v' \wedge (e \mapsto v'')) \ * \ (r/v' \to v)\}\\
\quad \left.\begin{array}{l}
\{(v = v' \wedge (e \mapsto v'')) \ * \ (r/v' \to v)\}\\
\quad \left.\begin{array}{l}
\{v = v' \wedge (e \mapsto v'')\}\\
v := [e]\\
\{v = v'' \wedge (e' \mapsto v)\}
\end{array}\right\} \ * \ r\\
\{(v = v'' \wedge (e' \mapsto v)) \ * \ (r/v'' \to v)\}
\end{array}\right\} \exists v', v''\\
\{\exists v', v''.\ (v = v'' \wedge (e' \mapsto v)) \ * \ (r/v'' \to v)\}\\
\{\exists v'.\ (e' \mapsto v) \ * \ (r/v'' \to v)\}.
\end{array}
$$

The global form (LKG) is the most commonly used of the rules that allow overwriting. As an example of an instance, if we take

| | | | | | |
|---|---|---|---|---|---|
| $v$ | to be | $\mathsf{j}$ | $e$ | to be | $\mathsf{j} + 1$ |
| $v'$ | to be | $\mathsf{m}$ | $r$ | to be | $\mathsf{i} + 1 \mapsto \mathsf{m} \ * \ \mathsf{k} + 1 \mapsto \mathbf{nil}$, |
| $v''$ | to be | $\mathsf{k}$ | | | |

then we obtain (with a little use of the commutivity of $*$)

$$
\begin{aligned}
&\{\exists \mathsf{k}.\ \mathsf{i}+1 \mapsto \mathsf{j} \ *\ \mathsf{j}+1 \mapsto \mathsf{k} \ *\ \mathsf{k}+1 \mapsto \mathbf{nil}\}\\
&\mathsf{j} := [\mathsf{j}+1]\\
&\{\exists \mathsf{m}.\ \mathsf{i}+1 \mapsto \mathsf{m} \ *\ \mathsf{m}+1 \mapsto \mathsf{j} \ *\ \mathsf{j}+1 \mapsto \mathbf{nil}\}.
\end{aligned}
$$

To derive (LKL) from (LKG), we first rename the variables $v'$ and $v''$ in (LKG) to be $\hat{v}'$ and $\hat{v}''$, chosen not to occur free in $e$, and then replace $r$ in (LKG) by $\hat{v}' = v' \wedge \hat{v}'' = v'' \wedge \mathbf{emp}$. Then we use the purity of equality, and predicate-calculus properties of equality and the existential quantifier, to obtain

$$
\begin{aligned}
&\{v = v' \wedge (e \mapsto v'')\}\\
&\{\exists \hat{v}''.\ v = v' \wedge \hat{v}'' = v'' \wedge (e \mapsto \hat{v}'')\}\\
&\{\exists \hat{v}''.\ (e \mapsto \hat{v}'') \ *\ (v = v' \wedge \hat{v}'' = v'' \wedge \mathbf{emp})\}\\
&\{\exists \hat{v}''.\ (e \mapsto \hat{v}'') \ *\ ((\hat{v}' = v' \wedge \hat{v}'' = v'' \wedge \mathbf{emp})/\hat{v}' \to v)\}\\
&v := [e]\\
&\{\exists \hat{v}'.\ (\hat{e}' \mapsto v) \ *\ ((\hat{v}' = v' \wedge \hat{v}'' = v'' \wedge \mathbf{emp})/\hat{v}'' \to v)\}\\
&\{\exists \hat{v}'.\ (\hat{e}' \mapsto v) \ *\ (\hat{v}' = v' \wedge v = v'' \wedge \mathbf{emp})\}\\
&\{\exists \hat{v}'.\ \hat{v}' = v' \wedge v = v'' \wedge (\hat{e}' \mapsto v)\}\\
&\{v = v'' \wedge (e' \mapsto v)\}
\end{aligned}
$$

(where $\hat{e}'$ denotes $e/v \to \hat{v}'$).

Turning to the backward-reasoning rules, we will derive (LKBR1) from (LKG). The reasoning here is tricky. We first derive a variant of (LKBR1) in which the variable $v''$ is renamed to a fresh variable $\hat{v}''$ that is distinct from $v$. Specifically, we assume $\hat{v}'' \neq v$, $\hat{v}'' \notin \mathrm{FV}(e) \cup (\mathrm{FV}(p) - \{v\})$, and $\hat{p}'' = p/v \to \hat{v}''$. We also take $v'$ to be a fresh variable, and take $r$ to be $(e' \mapsto \hat{v}'') \mathrel{-\!\!*} \hat{p}''$, where $e' = e/v \to v'$. Then, using (LKG) and the axiom schema (2.5) $q \ *\ (q \mathrel{-\!\!*} p) \Rightarrow p$, we obtain

$$
\begin{aligned}
&\{\exists \hat{v}''.\ (e \mapsto \hat{v}'') \ *\ ((e \mapsto \hat{v}'') \mathrel{-\!\!*} \hat{p}'')\}\\
&v := [e]\\
&\{\exists v'.\ (e' \mapsto v) \ *\ ((e' \mapsto v) \mathrel{-\!\!*} p)\}\\
&\{\exists v'.\ p\}\\
&\{p\},
\end{aligned}
$$

Now we consider (LKBR1) itself. The side condition $v'' \notin \mathrm{FV}(e) \cup (\mathrm{FV}(p) - \{v\})$ implies $v'' \notin \mathrm{FV}(e)$ and, since $\hat{v}''$ is fresh, $v'' \notin \hat{p}''$. This allows us to rename $\hat{v}''$ to $v''$ in the first line of the proof, to obtain a proof of (LKBR1).

To derive (LKBR2) from (LKBR1), we use the last axiom schema in (2.4): $(e \hookrightarrow e') \wedge p \Rightarrow (e \mapsto e') \ast ((e \mapsto e') \mathrel{-\!\!\ast} p)$.

$$\begin{array}{l}
\{\exists v''.\ (e \hookrightarrow v'') \wedge p''\} \\
\{\exists v''.\ (e \mapsto v'') \ast ((e \mapsto v'') \mathrel{-\!\!\ast} p'')\} \\
v := [e] \\
\{p\}.
\end{array}$$

Then to derive (LKL) from (LKBR2), we rename $v''$ to $\hat{v}$ in the precondition of (LKBR2), take $p$ to be $v = v'' \wedge (e' \mapsto v)$, and use properties of $\hookrightarrow$, equality, and the existential quantifier:

$$\begin{array}{l}
\{v = v' \wedge (e \mapsto v'')\} \\
\{v = v' \wedge (e \hookrightarrow v'') \wedge (e' \mapsto v'')\} \\
\{(e \hookrightarrow v'') \wedge (e' \mapsto v'')\} \\
\{\exists \hat{v}.\ (e \hookrightarrow v'') \wedge \hat{v} = v'' \wedge (e' \mapsto v'')\} \\
\{\exists \hat{v}.\ (e \hookrightarrow \hat{v}) \wedge \hat{v} = v'' \wedge (e' \mapsto \hat{v})\} \\
v := [e] \\
\{v = v'' \wedge (e' \mapsto v)\}.
\end{array}$$

The reader may verify that we have given more than enough derivations to establish that all of the rules for lookup that permit overwriting are interderivable.

For good measure, we also derive the global nonoverwriting rule (LKNOG) from (LKG): Suppose $v$ and $v''$ are distinct variables, $v \notin \mathrm{FV}(e)$, and $v'' \notin \mathrm{FV}(e) \cup \mathrm{FV}(p)$. We take $v'$ to be a variable distinct from $v$ and $v''$ that does not occur free in $e$ or $p$, and $r$ to be $p'' = p/v \to v''$. (Note that $v \notin \mathrm{FV}(e)$

implies that $e' = e$.) Then

$$\begin{array}{l}\{\exists v''.\ (e \mapsto v'')\ *\ p''\}\\ \{\exists v''.\ (e \mapsto v'')\ *\ (p''/v' \to v)\}\\ v := [e]\\ \{\exists v'.\ (e' \mapsto v)\ *\ (p''/v'' \to v)\}\\ \{\exists v'.\ (e \mapsto v)\ *\ p\}\\ \{(e \mapsto v)\ *\ p\}.\end{array}$$

In the first line, we can rename the quantified variable $v''$ to be any variable not in $\mathrm{FV}(e) \cup (\mathrm{FV}(p) - \{v\})$, so that $v''$ does not need to be distinct from $v$.

## 3.10 Annotated Specifications for the New Rules

It is straightforward to give annotated specifications for the backward-reasoning rules for mutation, disposal, allocation, and lookup; as with all backward-reasoning rules, these annotated specifications do not have explicit preconditions. Thus (MUBR), (DISBR), (CONSBR), and (LKBR1) lead to:

- Mutation (MUBRan)

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

- Disposal (DISBRan)

$$\frac{}{\mathbf{dispose}\ e\,\{r\} \gg \{(e \mapsto -)\ *\ r\}\ \mathbf{dispose}\ e\ \{r\}.}$$

- Allocation (CONSBRan)

$$\frac{}{v := \mathbf{cons}(\overline{e})\,\{p\} \gg \{\forall v''.\ (v'' \mapsto \overline{e}) \mathrel{-\!\!*} p''\}\ v := \mathbf{cons}(\overline{e})\ \{p\},}$$

  where $v''$ is distinct from $v$, $v'' \notin \mathrm{FV}(\overline{e}, p)$, and $p''$ denotes $p/v \to v''$.

- Lookup (LKBR1an)

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

  where $v'' \notin \mathrm{FV}(e) \cup (\mathrm{FV}(p) - \{v\})$, and $p''$ denotes $p/v \to v''$.

Moreover, since the implicit preconditions of these rules are weakest preconditions, the rules can be used to derive annotations (with explicit preconditions) for the other forms of the heap-manipulating rules. For example, by taking $p$ in (MUBRan) to be $e \mapsto e'$, and using the valid verification condition

$$\mathrm{VC} = (e \mapsto -) \Rightarrow (e \mapsto -) \ast ((e \mapsto e') \mathrel{-\!\!\ast} (e \mapsto e')),$$

we may use (SPan) to obtain a proof

$$\frac{\mathrm{VC} \qquad \dfrac{}{[e] := e' \, \{e \mapsto e'\} \gg \{(e \mapsto -) \ast ((e \mapsto e') \mathrel{-\!\!\ast} (e \mapsto e'))\} \; [e] := e' \; \{e \mapsto e'\}}}{\{e \mapsto -\} \; [e] := e' \; \{e \mapsto e'\} \gg \{e \mapsto -\} \; [e] := e' \; \{e \mapsto e'\}}$$

of an annotation description corresponding to the local form (MUL).

In such a manner, one may derive rules of the form

$$\frac{}{\{p\} \; c \; \{q\} \gg \{p\} \; c \; \{q\}}$$

corresponding to each of rules (MUL), (MUG), (DISL), (CONSNOL), (CONSNOG), (CONSL), (CONSG), (LKNOL), (LKNOG), (LKL), (LKG), and (LKBR2).

## 3.11 A Final Example

In conclusion, we reprise the annotated specification given at the end of Section 1.5, this time indicating the particular inference rules and verification conditions that are used. (To make the action of the inference rules clear, we have also given the unabbreviated form of each assertion.)

$$\{\mathbf{emp}\}$$

$$\mathsf{x} := \mathbf{cons}(\mathsf{a},\mathsf{a})\ ; \qquad \text{(CONSNOL)}$$

$$\{\mathsf{x} \mapsto \mathsf{a},\mathsf{a}\} \quad \text{i.e., } \{\mathsf{x} \mapsto \mathsf{a} \ * \ \mathsf{x}+1 \mapsto \mathsf{a}\}$$

$$\mathsf{y} := \mathbf{cons}(\mathsf{b},\mathsf{b})\ ; \qquad \text{(CONSNOG)}$$

$$\{(\mathsf{x} \mapsto \mathsf{a},\mathsf{a}) \ * \ (\mathsf{y} \mapsto \mathsf{b},\mathsf{b})\}$$
$$\text{i.e., } \{\mathsf{x} \mapsto \mathsf{a} \ * \ \mathsf{x}+1 \mapsto \mathsf{a} \ * \ \mathsf{y} \mapsto \mathsf{b} \ * \ \mathsf{y}+1 \mapsto \mathsf{b}\}$$

$$\{(\mathsf{x} \mapsto \mathsf{a},-) \ * \ (\mathsf{y} \mapsto \mathsf{b},\mathsf{b})\} \qquad (p/v \to e \Rightarrow \exists v.\ p)$$
$$\text{i.e., } \{\mathsf{x} \mapsto \mathsf{a} \ * \ (\exists \mathsf{a}.\ \mathsf{x}+1 \mapsto \mathsf{a}) \ * \ \mathsf{y} \mapsto \mathsf{b} \ * \ \mathsf{y}+1 \mapsto \mathsf{b}\}$$

$$[\mathsf{x}+1] := \mathsf{y}-\mathsf{x}\ ; \qquad \text{(MUG)}$$

$$\{(\mathsf{x} \mapsto \mathsf{a},\mathsf{y}-\mathsf{x}) \ * \ (\mathsf{y} \mapsto \mathsf{b},\mathsf{b})\}$$
$$\text{i.e., } \{\mathsf{x} \mapsto \mathsf{a} \ * \ \mathsf{x}+1 \mapsto \mathsf{y}-\mathsf{x} \ * \ \mathsf{y} \mapsto \mathsf{b} \ * \ \mathsf{y}+1 \mapsto \mathsf{b}\}$$

$$\{(\mathsf{x} \mapsto \mathsf{a},\mathsf{y}-\mathsf{x}) \ * \ (\mathsf{y} \mapsto \mathsf{b},-)\} \qquad (p/v \to e \Rightarrow \exists v.\ p)$$
$$\text{i.e., } \{\mathsf{x} \mapsto \mathsf{a} \ * \ \mathsf{x}+1 \mapsto \mathsf{y}-\mathsf{x} \ * \ \mathsf{y} \mapsto \mathsf{b} \ * \ (\exists \mathsf{b}.\ \mathsf{y}+1 \mapsto \mathsf{b})\}$$

$$[\mathsf{y}+1] := \mathsf{x}-\mathsf{y}\ ; \qquad \text{(MUG)}$$

$$\{(\mathsf{x} \mapsto \mathsf{a},\mathsf{y}-\mathsf{x}) \ * \ (\mathsf{y} \mapsto \mathsf{b},\mathsf{x}-\mathsf{y})\}$$
$$\text{i.e., } \{\mathsf{x} \mapsto \mathsf{a} \ * \ \mathsf{x}+1 \mapsto \mathsf{y}-\mathsf{x} \ * \ \mathsf{y} \mapsto \mathsf{b} \ * \ \mathsf{y}+1 \mapsto \mathsf{x}-\mathsf{y}\}$$

$$\{(\mathsf{x} \mapsto \mathsf{a},\mathsf{y}-\mathsf{x}) \ * \ (\mathsf{y} \mapsto \mathsf{b},-(\mathsf{y}-\mathsf{x}))\} \qquad (x-y=-(y-x))$$
$$\text{i.e., } \{\mathsf{x} \mapsto \mathsf{a} \ * \ \mathsf{x}+1 \mapsto \mathsf{y}-\mathsf{x} \ * \ \mathsf{y} \mapsto \mathsf{b} \ * \ \mathsf{y}+1 \mapsto -(\mathsf{y}-\mathsf{x})\}$$

$$\{(\mathsf{x} \mapsto \mathsf{a},\mathsf{y}-\mathsf{x}) \ * \ (\mathsf{x}+(\mathsf{y}-\mathsf{x}) \mapsto \mathsf{b},-(\mathsf{y}-\mathsf{x}))\} \quad (y=x+(y-x))$$
$$\text{i.e., } \{\mathsf{x} \mapsto \mathsf{a} \ * \ \mathsf{x}+1 \mapsto \mathsf{y}-\mathsf{x}$$
$$* \ \mathsf{x}+(\mathsf{y}-\mathsf{x}) \mapsto \mathsf{b} \ * \ \mathsf{x}+(\mathsf{y}-\mathsf{x})+1 \mapsto -(\mathsf{y}-\mathsf{x})\}$$

$$\{\exists \mathsf{o}.\ (\mathsf{x} \mapsto \mathsf{a},\mathsf{o}) \ * \ (\mathsf{x}+\mathsf{o} \mapsto \mathsf{b},\ -\mathsf{o})\} \qquad (p/v \to e \Rightarrow \exists v.\ p)$$
$$\text{i.e., } \{\mathsf{x} \mapsto \mathsf{a} \ * \ \mathsf{x}+1 \mapsto \mathsf{o} \ * \ \mathsf{x}+\mathsf{o} \mapsto \mathsf{b} \ * \ \mathsf{x}+\mathsf{o}+1 \mapsto -\mathsf{o}\}$$

## 3.12 More about Annotated Specifications

In this section, we will show the essential property of annotated specifications: From a formal proof of a specification one can derive an annotated version of the specification, from which one can reconstruct a similar (though perhaps not identical) formal proof.

We begin by defining functions that map annotation descriptions into their underlying specifications and annotated specifications:

$$\text{erase-annspec}(\mathcal{A} \gg \{p\}\ c\ \{q\}) \stackrel{\text{def}}{=} \{p\}\ c\ \{q\}$$

$$\text{erase-spec}(\mathcal{A} \gg \{p\}\ c\ \{q\}) \stackrel{\text{def}}{=} \mathcal{A}.$$

We extend these functions to inference rules and proofs of annotation descriptions, in which case they are applied to all of the annotation descriptions in the inference rule or proof (while leaving verification conditions unchanged).

We also define a function "concl" that maps proofs (of either specifications or annotation descriptions) into their conclusions.

Then we define a function cd that acts on an annotated specification $\mathcal{A}$ by deleting annotations to produce the command imbedded within $\mathcal{A}$. For most forms of annotation, the definition is obvious, but in the case of DISJ and CONJ, it relies on the fact that the commands imbedded in the subannotations must be identical:

$$\text{cd}(\mathcal{A}_1\,\text{DISJ}\,\mathcal{A}_2) = \text{cd}(\mathcal{A}_1) = \text{cd}(\mathcal{A}_2)$$

$$\text{cd}(\mathcal{A}_1\,\text{CONJ}\,\mathcal{A}_2) = \text{cd}(\mathcal{A}_1) = \text{cd}(\mathcal{A}_2).$$

In the case of an annotation for the substitution rule, the substitution is carried out:

$$\text{cd}(\Big\{\mathcal{A}\Big\}/\delta) = \text{cd}(\mathcal{A})/\delta.$$

**Proposition 12** *1. The function* erase-annspec *maps each inference rule* (Xan) *into the rule* (X). *(We are ignoring the alternative rules* (ASalt), (CDalt)*, and* (CONSEQ).*)*

2. *The function* erase-annspec *maps proofs of annotation descriptions into proofs of specifications.*

3. *Suppose $\mathcal{A} \gg \{p\}\ c\ \{q\}$ is a provable annotation description. Then* $\text{cd}(\mathcal{A}) = c$. *Moreover, if $\mathcal{A}$ is left-complete, then it begins with $\{p\}$, and if $\mathcal{A}$ is right-complete then it ends with $\{q\}$.*

PROOF The proof of (1) is by case analysis on the individual pairs of rules. The proofs of (2) and (3) are straightforward inductions on the structure of proofs of annotation descriptions. END OF PROOF

We use $\mathcal{P}$ or $\mathcal{Q}$ (with occasional decorations) as variables that range over proofs of specifications or of annotation descriptions respectively. We also use the notations $\mathcal{P}[\{p\}\ c\ \{q\}]$ or $\mathcal{Q}[\mathcal{A} \gg \{p\}\ c\ \{q\}]$ as variables whose range is limited to proofs with a particular conclusion.

Next, we introduce three endofunctions on proofs of annotation descriptions that force annotations to be complete by introducing vacuous instances of the rules (SP) and (WC). If $\mathcal{A}$ is left-complete, then

$$\text{left-compl}(\mathcal{Q}[\mathcal{A} \gg \{p\}\ c\ \{q\}]) = \mathcal{Q}[\mathcal{A} \gg \{p\}\ c\ \{q\}],$$

otherwise,

$$\text{left-compl}(\mathcal{Q}[\mathcal{A} \gg \{p\}\ c\ \{q\}]) = \frac{p \Rightarrow p \quad \mathcal{Q}[\mathcal{A} \gg \{p\}\ c\ \{q\}]}{\{p\}\mathcal{A} \gg \{p\}\ c\ \{q\}.}$$

Similarly, if $\mathcal{A}$ is right-complete, then

$$\text{right-compl}(\mathcal{Q}[\mathcal{A} \gg \{p\}\ c\ \{q\}]) = \mathcal{Q}[\mathcal{A} \gg \{p\}\ c\ \{q\}],$$

otherwise,

$$\text{right-compl}(\mathcal{Q}[\mathcal{A} \gg \{p\}\ c\ \{q\}]) = \frac{\mathcal{Q}[\mathcal{A} \gg \{p\}\ c\ \{q\}] \quad q \Rightarrow q}{\mathcal{A}\{q\} \gg \{p\}\ c\ \{q\}.}$$

Then, combining these functions,

$$\begin{aligned}&\text{compl}(\mathcal{Q}[\mathcal{A} \gg \{p\}\ c\ \{q\}]) = \\ &\qquad\qquad \text{left-compl}(\text{right-compl}(\mathcal{Q}[\mathcal{A} \gg \{p\}\ c\ \{q\}])).\end{aligned}$$

Now we can define a function $\Phi$, mapping proofs of specifications into proofs of annotation descriptions, that satisfies

**Proposition 13** *1. If $\mathcal{P}$ proves $\{p\}\ c\ \{q\}$, then $\Phi(\mathcal{P})$ proves $\mathcal{A} \gg \{p\}\ c\ \{q\}$ for some annotated specification $\mathcal{A}$.*

*2.* erase-annspec$(\Phi(\mathcal{P}))$ *is similar to $\mathcal{P}$, except for the possible insertion of instances of* (SP) *and* (WC) *in which the verification condition is a trivial implication of the form $p \Rightarrow p$.*

Note that Part 2 of this proposition implies that, except for extra implications of the form $p \Rightarrow p$, erase-annspec($\Phi(\mathcal{P})$) contains the same verification conditions as $\mathcal{P}$.

We define $\Phi(\mathcal{P})$ by induction on the structure of $\mathcal{P}$, with a case analysis on the final rule used to infer the conclusion of $\mathcal{P}$. The above proposition is proved concurrently (and straightforwardly) by the same induction and case analysis.

(AS) If $\mathcal{P}$ is

$$\frac{}{\{q/v \rightarrow e\}\ v := e\ \{q\},}$$

then $\Phi(\mathcal{P})$ is

$$\frac{}{v := e\,\{q\} \gg \{q/v \rightarrow e\}\ v := e\ \{q\}.}$$

(SP) If $\mathcal{P}$ is

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

and

$$\Phi(\mathcal{P}'[\{q\}\ c\ \{r\}]) = \mathcal{Q}'[\mathcal{A} \gg \{q\}\ c\ \{r\}],$$

then $\Phi(\mathcal{P})$ is

$$\frac{p \Rightarrow q \qquad \mathcal{Q}'[\mathcal{A} \gg \{q\}\ c\ \{r\}]}{\{p\}\,\mathcal{A} \gg \{p\}\ c\ \{r\}.}$$

(WC) If $\mathcal{P}$ is

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

and

$$\Phi(\mathcal{P}'[\{p\}\ c\ \{q\}]) = \mathcal{Q}'[\mathcal{A} \gg \{p\}\ c\ \{q\}],$$

then $\Phi(\mathcal{P})$ is

$$\frac{\mathcal{Q}'[\mathcal{A} \gg \{p\}\ c\ \{q\}] \qquad q \Rightarrow r}{\mathcal{A}\,\{r\} \gg \{p\}\ c\ \{r\}.}$$

(SQ) If $\mathcal{P}$ is

$$\frac{\mathcal{P}_1[\{p\}\ c_1\ \{q\}] \qquad \mathcal{P}_2[\{q\}\ c_2\ \{r\}]}{\{p\}\ c_1\,;c_2\ \{r\},}$$

and

$$\text{right-compl}(\Phi(\mathcal{P}_1[\{p\}\ c_1\ \{q\}])) = \mathcal{Q}_1[\mathcal{A}_1\{q\} \gg \{p\}\ c_1\ \{q\}]$$

$$\Phi(\mathcal{P}_2[\{q\}\ c_2\ \{r\}]) = \mathcal{Q}_2[\mathcal{A}_2 \gg \{q\}\ c_2\ \{r\}],$$

then $\Phi(\mathcal{P})$ is

$$\frac{\mathcal{Q}_1[\mathcal{A}_1\,\{q\} \gg \{p\}\ c_1\ \{q\}] \qquad \mathcal{Q}_2[\mathcal{A}_2 \gg \{q\}\ c_2\ \{r\}]}{\mathcal{A}_1\,;\mathcal{A}_2 \gg \{p\}\ c_1\,;c_2\ \{r\}.}$$

(WH) If $\mathcal{P}$ is

$$\frac{\mathcal{P}'[\{i \wedge b\}\ c\ \{i\}]}{\{i\}\ \mathbf{while}\ b\ \mathbf{do}\ c\ \{i \wedge \neg\, b\},}$$

and

$$\text{compl}(\Phi(\mathcal{P}'[\{i \wedge b\}\ c\ \{i\}])) = \mathcal{Q}'[\{i \wedge b\}\ \mathcal{A}\ \{i\} \gg \{i \wedge b\}\ c\ \{i\}],$$

then $\Phi(\mathcal{P})$ is

$$\frac{\mathcal{Q}'[\{i \wedge b\}\ \mathcal{A}\ \{i\} \gg \{i \wedge b\}\ c\ \{i\}]}{\{i\}\,\mathbf{while}\ b\ \mathbf{do}\ (\mathcal{A}) \gg \{i\}\ \mathbf{while}\ b\ \mathbf{do}\ c\ \{i \wedge \neg\, b\}.}$$

(SK) If $\mathcal{P}$ is

$$\frac{}{\{q\}\ \mathbf{skip}\ \{q\},}$$

then $\Phi(\mathcal{P})$ is

$$\frac{}{\mathbf{skip}\,\{q\} \gg \{q\}\ \mathbf{skip}\ \{q\}.}$$

(CD) If $\mathcal{P}$ is

$$\frac{\mathcal{P}_1[\{p \wedge b\}\ c_1\ \{q\}] \qquad \mathcal{P}_2[\{p \wedge \neg\ b\}\ c_2\ \{q\}]}{\{p\}\ \mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2\ \{q\},}$$

and

$$\text{compl}(\Phi(\mathcal{P}_1[\{p \wedge b\}\ c_1\ \{q\}])) = \mathcal{Q}_1[\{p \wedge b\}\ \mathcal{A}_1\ \{q\} \gg \{p \wedge b\}\ c_1\ \{q\}]$$

$$\text{compl}(\Phi(\mathcal{P}_2[\{p \wedge \neg\, b\}\ c_2\ \{q\}])) =$$
$$\mathcal{Q}_2[\{p \wedge \neg\, b\}\ \mathcal{A}_2\ \{q\} \gg \{p \wedge \neg\, b\}\ c_2\ \{q\}],$$

then $\Phi(\mathcal{P})$ is

$$\frac{\begin{array}{c}\mathcal{Q}_1[\{p \wedge b\}\ \mathcal{A}_1\ \{q\} \gg \{p \wedge b\}\ c_1\ \{q\}] \\ \mathcal{Q}_2[\{p \wedge \neg\, b\}\ \mathcal{A}_2\ \{q\} \gg \{p \wedge \neg\, b\}\ c_2\ \{q\}]\end{array}}{\{p\}\ \textbf{if}\ b\ \textbf{then}\ \mathcal{A}_1\ \textbf{else}\ (\mathcal{A}_2)\ \{q\} \gg \{p\}\ \textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c_2\ \{q\}.}$$

(DC) If $\mathcal{P}$ is

$$\frac{\mathcal{P}'[\{p\}\ c\ \{q\}]}{\{p\}\ \textbf{newvar}\ v\ \textbf{in}\ c\ \{q\},}$$

when $v$ does not occur free in $p$ or $q$, and

$$\text{compl}(\Phi(\mathcal{P}'[\{p\}\ c\ \{q\}])) = \mathcal{Q}'[\{p\}\ \mathcal{A}\ \{q\} \gg \{p\}\ c\ \{q\}],$$

then $\Phi(\mathcal{P})$ is

$$\frac{\mathcal{Q}'[\{p\}\ \mathcal{A}\ \{q\} \gg \{p\}\ c\ \{q\}]}{\{p\}\ \textbf{newvar}\ v\ \textbf{in}\ (\mathcal{A})\ \{q\} \gg \{p\}\ \textbf{newvar}\ v\ \textbf{in}\ c\ \{q\}.}$$

(VAC) If $\mathcal{P}$ is

$$\frac{}{\{\textbf{false}\}\ c\ \{q\},}$$

then $\Phi(\mathcal{P})$ is

$$\frac{}{\Big\{c\Big\}\mathrm{VAC}\,\{q\} \gg \{\textbf{false}\}\ c\ \{q\}.}$$

(DISJ) If $\mathcal{P}$ is

$$\frac{\mathcal{P}_1[\{p_1\}\ c\ \{q\}] \qquad \mathcal{P}_2[\{p_2\}\ c\ \{q\}]}{\{p_1 \vee p_2\}\ c\ \{q\},}$$

and

$$\text{right-compl}(\Phi(\mathcal{P}_1[\{p_1\}\ c\ \{q\}])) = \mathcal{Q}_1[\mathcal{A}_1\,\{q\} \gg \{p_1\}\ c\ \{q\}]$$

$$\text{right-compl}(\Phi(\mathcal{P}_2[\{p_2\}\ c\ \{q\}])) = \mathcal{Q}_2[\mathcal{A}_2\,\{q\} \gg \{p_2\}\ c\ \{q\}],$$

then $\Phi(\mathcal{P})$ is

$$\frac{\mathcal{Q}_1[\mathcal{A}_1\,\{q\} \gg \{p_1\}\ c\ \{q\}] \qquad \mathcal{Q}_2[\mathcal{A}_2\,\{q\} \gg \{p_2\}\ c\ \{q\}]}{(\mathcal{A}_1\ \mathit{DISJ}\ \mathcal{A}_2)\,\{q\} \gg \{p_1 \vee p_2\}\ c\ \{q\}.}$$

(CONJ) If $\mathcal{P}$ is

$$\frac{\mathcal{P}_1[\{p_1\}\ c\ \{q_1\}] \qquad \mathcal{P}_2[\{p_2\}\ c\ \{q_2\}]}{\{p_1 \wedge p_2\}\ c\ \{q_1 \wedge q_2\},}$$

and

$$\Phi(\mathcal{P}_1[\{p_1\}\ c\ \{q_1\}]) = \mathcal{Q}_1[\mathcal{A}_1 \gg \{p_1\}\ c\ \{q_1\}]$$

$$\Phi(\mathcal{P}_2[\{p_2\}\ c\ \{q_2\}]) = \mathcal{Q}_2[\mathcal{A}_2 \gg \{p_2\}\ c\ \{q_2\}],$$

then $\Phi(\mathcal{P})$ is

$$\frac{\mathcal{Q}_1[\mathcal{A}_1 \gg \{p_1\}\ c\ \{q_1\}] \qquad \mathcal{Q}_2[\mathcal{A}_2 \gg \{p_2\}\ c\ \{q_2\}]}{(\mathcal{A}_1\ CONJ\ \mathcal{A}_2) \gg \{p_1 \wedge p_2\}\ c\ \{q_1 \wedge q_2\}.}$$

(EQ) If $\mathcal{P}$ is

$$\frac{\mathcal{P}'[\{p\}\ c\ \{q\}]}{\{\exists v.\ p\}\ c\ \{\exists v.\ q\},}$$

where $v$ is not free in $c$, and

$$\Phi(\mathcal{P}'[\{p\}\ c\ \{q\}]) = \mathcal{Q}'[\mathcal{A} \gg \{p\}\ c\ \{q\}],$$

then $\Phi(\mathcal{P})$ is

$$\frac{\mathcal{Q}'[\mathcal{A} \gg \{p\}\ c\ \{q\}]}{\Big\{\mathcal{A}\Big\}\ \exists v\ \gg \{\exists v.\ p\}\ c\ \{\exists v.\ q\}.}$$

(UQ) is similar to (EQ).

(FR) If $\mathcal{P}$ is

$$\frac{\mathcal{P}'[\{p\}\ c\ \{q\}]}{\{p\ *\ r\}\ c\ \{q\ *\ r\},}$$

where no variable occurring free in $r$ is modified by $c$, and

$$\Phi(\mathcal{P}'[\{p\}\ c\ \{q\}]) = \mathcal{Q}'[\mathcal{A} \gg \{p\}\ c\ \{q\}],$$

then $\Phi(\mathcal{P})$ is

$$\frac{\mathcal{Q}'[\mathcal{A} \gg \{p\}\ c\ \{q\}]}{\Big\{\mathcal{A}\Big\}\ *\ r\ \gg \{p\ *\ r\}\ c\ \{q\ *\ r\}.}$$

(SUB) If $\mathcal{P}$ is

$$\frac{\mathcal{P}'[\{p\}\ c\ \{q\}]}{\{p/\delta\}\ (c/\delta)\ \{q/\delta\},}$$

where $\delta$ is the substitution $v_1 \to e_1, \ldots, v_n \to e_n$, $v_1, \ldots, v_n$ are the variables occurring free in $p$, $c$, or $q$, and, if $v_i$ is modified by $c$, then $e_i$ is a variable that does not occur free in any other $e_j$, and

$$\Phi(\mathcal{P}'[\{p\}\ c\ \{q\}]) = \mathcal{Q}'[\mathcal{A} \gg \{p\}\ c\ \{q\}],$$

then $\Phi(\mathcal{P})$ is

$$\frac{\mathcal{Q}'[\mathcal{A} \gg \{p\}\ c\ \{q\}]}{\Big\{\mathcal{A}\Big\}/\delta \gg \{p/\delta\}\ (c/\delta)\ \{q/\delta\}.}$$

(MUBR) If $\mathcal{P}$ is

$$\frac{}{\{(e \mapsto -) \ast ((e \mapsto e') \mathrel{-\!\!\ast} q)\}\ [e] := e'\ \{q\},}$$

then $\Phi(\mathcal{P})$ is

$$\frac{}{[e] := e'\ \{q\} \gg \{(e \mapsto -) \ast ((e \mapsto e') \mathrel{-\!\!\ast} q)\}\ [e] := e'\ \{q\}.}$$

(DISBR) If $\mathcal{P}$ is

$$\frac{}{\{(e \mapsto -) \ast q\}\ \mathbf{dispose}\ e\ \{q\},}$$

then $\Phi(\mathcal{P})$ is

$$\frac{}{\mathbf{dispose}\ e\ \{q\} \gg \{(e \mapsto -) \ast q\}\ \mathbf{dispose}\ e\ \{q\}.}$$

(CONSBR) If $\mathcal{P}$ is

$$\frac{}{\{\forall v''.\ (v'' \mapsto \overline{e}) \mathrel{-\!\!\ast} q''\}\ v := \mathbf{cons}(\overline{e})\ \{q\},}$$

where $v''$ is distinct from $v$, $v'' \notin \mathrm{FV}(\overline{e}, q)$, and $q''$ denotes $q/v \to v''$, then $\Phi(\mathcal{P})$ is

$$\frac{}{v := \mathbf{cons}(\overline{e})\ \{q\} \gg \{\forall v''.\ (v'' \mapsto \overline{e}) \mathrel{-\!\!\ast} q''\}\ v := \mathbf{cons}(\overline{e})\ \{q\}.}$$

(LKBR1) If $\mathcal{P}$ is

$$\frac{}{\{\exists v''.\ (e \mapsto v'') \ast ((e \mapsto v'') \mathrel{-\!\!\ast} q'')\}\ v := [e]\ \{q\},}$$

where $v'' \notin \mathrm{FV}(e) \cup (\mathrm{FV}(q) - \{v\})$, and $q''$ denotes $q/v \to v''$, then $\Phi(\mathcal{P})$ is

$$\frac{}{v := [e]\ \{q\} \gg \{\exists v''.\ (e \mapsto v'') \ast ((e \mapsto v'') \mathrel{-\!\!\ast} q'')\}\ v := [e]\ \{q\}.}$$

Finally, we will close the circle by defining a function $\Psi$ that maps annotated specifications into proofs of the specifications that they annotate. But first, we must define certain sequences of premisses and proofs, and associated concepts:

- A *premiss from $p$ to $q$* is either a specification $\{p\}\ c\ \{q\}$ or a verification condition $p \Rightarrow q$.

- A *coherent premiss sequence from $p$ to $q$* is either a premiss from $p$ to $q$, or a shorter coherent premiss sequence from $p$ to some $r$, followed by a premiss from $r$ to $q$. We use $\mathcal{S}$ (with occasional decorations) as a variable that ranges over coherent premiss sequences, and $\mathcal{S}[p, q]$ as a variable that ranges over coherent premiss sequences from $p$ to $q$.

- A *proof from $p$ to $q$* is either a proof of a specification $\{p\}\ c\ \{q\}$ or a verification condition $p \Rightarrow q$.

- A *coherent proof sequence from $p$ to $q$* is either a proof from $p$ to $q$, or a shorter coherent proof sequence from $p$ to some $r$, followed by a premiss from $r$ to $q$. We use $\mathcal{R}$ (with occasional decorations) as a variable that ranges over coherent proof sequences, and $\mathcal{R}[p, q]$ as a variable that ranges over coherent proof sequences from $p$ to $q$.

- A coherent premiss (or proof) sequence is *proper* if it contains at least one specification (or proof of a specification).

- We extend the function "concl" to map coherent proof sequences into coherent premiss sequences by replacing each proof of a specification by its conclusion, and leaving verification conditions unchanged.

- The function "code" maps proper coherent premiss sequences into commands by sequentially composing the commands occurring within the premisses that are specifications. More precisely (where code$'$ is an auxilliary function mapping coherent premiss sequences or the empty sequence $\epsilon$ into commands):

$$\begin{aligned}
\mathrm{code}'(\mathcal{S}, \{p\}\ c\ \{q\}) &= \mathrm{code}'(\mathcal{S})\ ;c \\
\mathrm{code}'(\mathcal{S}, p \Rightarrow q) &= \mathrm{code}'(\mathcal{S}) \\
\mathrm{code}'(\epsilon) &= \mathbf{skip} \\
\mathrm{code}(\mathcal{S}) &= c \text{ when } \mathrm{code}'(\mathcal{S}) = \mathbf{skip}\ ;c.
\end{aligned}$$

The utility of these concepts is that, for any proper coherent premiss sequence $\mathcal{S}$ from $p$ to $q$, one can derive the inference rule

$$\frac{\mathcal{S}}{\{p\}\ \mathrm{code}(\mathcal{S})\ \{q\}.}$$

The derivation is obtained by using the rules (SQ), (SP), and (WC) to build up a proper coherent proof sequence $\mathcal{R}$ from $p$ to $q$ in which the components of $\mathcal{S}$ occur as assumptions. One begins by taking $\mathcal{R}$ to be $\mathcal{S}$, and then repeatedly applies the following nondeterminate step:

> If $\mathcal{R}$ can be put in the form
>
> $$\mathcal{R}_1, \mathcal{P}_1[\{p'\}\ c_1\ \{q'\}], \mathcal{P}_2[\{q'\}\ c_2\ \{r'\}], \mathcal{R}_2,$$
>
> replace it by
>
> $$\mathcal{R}_1, \frac{\mathcal{P}_1[\{p'\}\ c_1\ \{q'\}] \quad \mathcal{P}_2[\{q'\}\ c_2\ \{r'\}]}{\{p'\}\ c_1\,;c_2\ \{r'\}}, \mathcal{R}_2,$$
>
> or, if $\mathcal{R}$ can be put in the form
>
> $$\mathcal{R}_1, p' \Rightarrow q', \mathcal{P}_2[\{q'\}\ c\ \{r'\}], \mathcal{R}_2,$$
>
> replace it by
>
> $$\mathcal{R}_1, \frac{p' \Rightarrow q' \quad \mathcal{P}_2[\{q'\}\ c\ \{r'\}]}{\{p'\}\ c\ \{r'\}}, \mathcal{R}_2,$$
>
> or if $\mathcal{R}$ can be put in the form
>
> $$\mathcal{R}_1, \mathcal{P}_1[\{p'\}\ c\ \{q'\}], q' \Rightarrow r', \mathcal{R}_2,$$
>
> replace it by
>
> $$\mathcal{R}_1, \frac{\mathcal{P}_1[\{p'\}\ c\ \{q'\}] \quad q' \Rightarrow r'}{\{p'\}\ c\ \{r'\}}, \mathcal{R}_2.$$

Each step reduces the length of $\mathcal{R}$ while preserving code(concl($\mathcal{R}$)), so that eventually one reaches the state where $\mathcal{R}$ is a single proof, of $\{p\}$ code($\mathcal{S}$) $\{q\}$.

We will define the function $\Psi$ in terms of a function $\Psi_0$ that maps annotations into proper coherent proof sequences. Specifically, if $\Psi_0(\mathcal{A})$ is a proper coherent proof sequence from $p$ to $q$, then

$$\Psi(\mathcal{A}) = \frac{\Psi_0(\mathcal{A})}{\{p\}\ \mathrm{code}(\mathrm{concl}(\Psi_0(\mathcal{A})))\ \{q\}.}$$

Strictly speaking, we should replace the final step of this proof by one of its derivations. The annotation $\mathcal{A}$ contains insufficient information to determine which of these derivations should be used; fortunately, the choice doesn't matter.

The function $\Psi_0$ is defined as follows:

$$
\begin{aligned}
\Psi_0(\{q\}) &= \epsilon \\
\Psi_0(\mathcal{A}_0\, \{p\}\, \{q\}) &= \Psi_0(\mathcal{A}_0\, \{p\}), p \Rightarrow q \\
\Psi_0(\mathcal{A}_0\, v := e\, \{q\}) &= \Psi_0(\mathcal{A}_0\, \{q/v \rightarrow e\}),\ \overline{\{q/v \rightarrow e\}\ v := e\ \{q\}} \\
\Psi_0(\mathcal{A}_0\, ;\, \{q\}) &= \Psi_0(\mathcal{A}_0\, \{q\}) \\
\Psi_0(\mathcal{A}_0\, \{i\}\, \mathbf{while}\ b\ \mathbf{do}\ (\mathcal{A})\, \{q\}) &= \Psi_0(\mathcal{A}_0\, \{i\}\, \mathbf{while}\ b\ \mathbf{do}\ (\mathcal{A}))\, \{i \wedge \neg\, b \Rightarrow q\} \\
\Psi_0(\mathcal{A}_0\, \{i\}\, \mathbf{while}\ b\ \mathbf{do}\ (\mathcal{A})) &= \Psi_0(\mathcal{A}_0\, \{i\}),\ \frac{\Psi(\{i \wedge b\}\ \mathcal{A}\ \{i\})}{\{i\}\ \mathbf{while}\ b\ \mathbf{do}\ \mathrm{cd}(\mathcal{A})\ \{i \wedge \neg\, b\}} \\
\Psi_0(\mathcal{A}_0\, \mathbf{skip}\, \{q\}) &= \Psi_0(\mathcal{A}_0\, \{q\}),\ \overline{\{q\}\ \mathbf{skip}\ \{q\}}
\end{aligned}
$$

$$
\begin{aligned}
&\Psi_0(\mathcal{A}_0\, \{p\}\ \mathbf{if}\ b\ \mathbf{then}\ \mathcal{A}_1\ \mathbf{else}\ (\mathcal{A}_2)\ \{q\}) = \\
&\qquad\qquad \Psi_0(\mathcal{A}_0\, \{p\}),\ \frac{\Psi(\{p \wedge b\}\ \mathcal{A}_1\ \{q\}) \quad \Psi(\{p \wedge \neg\, b\}\ \mathcal{A}_2\ \{q\})}{\{p\}\ \mathbf{if}\ b\ \mathbf{then}\ \mathrm{cd}(\mathcal{A}_1)\ \mathbf{else}\ \mathrm{cd}(\mathcal{A}_2)\ \{q\}}
\end{aligned}
$$

$$
\begin{aligned}
\Psi_0(\mathcal{A}_0\, \{p\}\ \mathbf{newvar}\ v\ \mathbf{in}\ (\mathcal{A})\ \{q\}) &= \Psi_0(\mathcal{A}_0\, \{p\}),\ \frac{\Psi(\{p\}\ \mathcal{A}\ \{q\})}{\{p\}\ \mathbf{newvar}\ v\ \mathbf{in}\ \mathrm{cd}(\mathcal{A})\ \{q\}} \\
\Psi_0(\mathcal{A}_0\, \Big\{c\Big\}\mathrm{VAC}\, \{q\}) &= \Psi_0(\mathcal{A}_0\, \{\mathbf{false}\}),\ \overline{\{\mathbf{false}\}\ c\ \{q\}} \\
\Psi_0(\mathcal{A}_0\, (\mathcal{A}_1\, \mathrm{DISJ}\, \mathcal{A}_2)\, \{q\}) &= \Psi_0(\mathcal{A}_0\, \{p_1 \vee p_2\}),\ \frac{\Psi(\mathcal{A}_1\, \{q\}) \quad \Psi(\mathcal{A}_2\, \{q\})}{\{p_1 \vee p_2\}\ c\ \{q\}} \\
&\qquad\qquad \text{where } \Psi(\mathcal{A}_1\, \{q\}) \text{ proves } \{p_1\}\ c\ \{q\} \\
&\qquad\qquad \text{and } \Psi(\mathcal{A}_2\, \{q\}) \text{ proves } \{p_2\}\ c\ \{q\} \\
\Psi_0(\mathcal{A}_0\, (\mathcal{A}_1\, \mathrm{CONJ}\, \mathcal{A}_2)\, \{q\}) &= \Psi_0(\mathcal{A}_0\, (\mathcal{A}_1\, \mathrm{CONJ}\, \mathcal{A}_2), q_1 \wedge q_2 \Rightarrow q \\
&\qquad\qquad \text{where } \Psi(\mathcal{A}_1) \text{ proves } \{p_1\}\ c\ \{q_1\} \\
&\qquad\qquad \text{and } \Psi(\mathcal{A}_2) \text{ proves } \{p_2\}\ c\ \{q_2\}
\end{aligned}
$$

$$\Psi_0(\mathcal{A}_0\,(\mathcal{A}_1\,\mathrm{CONJ}\,\mathcal{A}_2)) = \Psi_0(\mathcal{A}_0\,\{p_1 \wedge p_2\}),\ \frac{\Psi(\mathcal{A}_1) \quad \Psi(\mathcal{A}_2)}{\{p_1 \wedge p_2\}\ c\ \{q_1 \wedge q_2\}}$$

where $\Psi(\mathcal{A}_1)$ proves $\{p_1\}\ c\ \{q_1\}$
and $\Psi(\mathcal{A}_2)$ proves $\{p_2\}\ c\ \{q_2\}$

$$\Psi_0(\mathcal{A}_0\,\Big\{\mathcal{A}\Big\}\,\exists v\,\{r\}) = \Psi_0(\mathcal{A}_0\,\Big\{\mathcal{A}\Big\}\,\exists v), q \Rightarrow r$$

where $\Psi(\mathcal{A})$ proves $\{p\}\ c\ \{q\}$

$$\Psi_0(\mathcal{A}_0\,\Big\{\mathcal{A}\Big\}\,\exists v) = \Psi_0(\mathcal{A}_0\,\{\exists v.\ p\}),\ \frac{\Psi(\mathcal{A})}{\{\exists v.\ p\}\ c\ \{\exists v.\ q\}}$$

where $\Psi(\mathcal{A})$ proves $\{p\}\ c\ \{q\}$

$$\Psi_0(\mathcal{A}_0\,\Big\{\mathcal{A}\Big\}\,\forall v\,\{r\}) = \Psi_0(\mathcal{A}_0\,\Big\{\mathcal{A}\Big\}\,\forall v), q \Rightarrow r$$

where $\Psi(\mathcal{A})$ proves $\{p\}\ c\ \{q\}$

$$\Psi_0(\mathcal{A}_0\,\Big\{\mathcal{A}\Big\}\,\forall v) = \Psi_0(\mathcal{A}_0\,\{\forall v.\ p\}),\ \frac{\Psi(\mathcal{A})}{\{\forall v.\ p\}\ c\ \{\forall v.\ q\}}$$

where $\Psi(\mathcal{A})$ proves $\{p\}\ c\ \{q\}$

$$\Psi_0(\mathcal{A}_0\,\Big\{\mathcal{A}\Big\}\, *\, r\,\{s\}) = \Psi_0(\mathcal{A}_0\,\Big\{\mathcal{A}\Big\}\, *\, r), q\, *\, r \Rightarrow s$$

where $\Psi(\mathcal{A})$ proves $\{p\}\ c\ \{q\}$

$$\Psi_0(\mathcal{A}_0\,\Big\{\mathcal{A}\Big\}\, *\, r) = \Psi_0(\mathcal{A}_0\,\{p\, *\, r\}),\ \frac{\Psi(\mathcal{A})}{\{p\, *\, r\}\ c\ \{q\, *\, r\}}$$

where $\Psi(\mathcal{A})$ proves $\{p\}\ c\ \{q\}$

$$\Psi_0(\mathcal{A}_0\,\Big\{\mathcal{A}\Big\}/\delta\,\{r\}) = \Psi_0(\mathcal{A}_0\,\Big\{\mathcal{A}\Big\}/\delta), q/\delta \Rightarrow r$$

where $\Psi(\mathcal{A})$ proves $\{p\}\ c\ \{q\}$

$$\Psi_0(\mathcal{A}_0\,\Big\{\mathcal{A}\Big\}/\delta) = \Psi_0(\mathcal{A}_0\,\{p/\delta\}),\ \frac{\Psi(\mathcal{A})}{\{p/\delta\}\ c\ \{q/\delta\}}$$

where $\Psi(\mathcal{A})$ proves $\{p\}\ c\ \{q\}$

$$\Psi_0(\mathcal{A}_0\,[e] := e'\,\{q\}) = \Psi_0(\mathcal{A}_0\,\{(e \mapsto -) \ast ((e \mapsto e') -\!\!\ast\, q)\}),$$

$$\frac{}{\{(e \mapsto -) \ast ((e \mapsto e') -\!\!\ast\, q)\}\ [e] := e'\ \{q\}}$$

$$\Psi_0(\mathcal{A}_0\,\mathbf{dispose}\ e\,\{q\}) = \Psi_0(\mathcal{A}_0\,\{(e \mapsto -) \ast q\}),$$

$$\frac{}{\{(e \mapsto -) \ast q\}\ \mathbf{dispose}\ e\ \{q\}}$$

$$\Psi_0(\mathcal{A}_0\,v := \mathbf{cons}(\overline{e})\,\{q\}) = \Psi_0(\mathcal{A}_0\,\{\forall v''.\ (v'' \mapsto \overline{e}) -\!\!\ast\, q''\}),$$

$$\frac{}{\{\forall v''.\ (v'' \mapsto \overline{e}) -\!\!\ast\, q''\}\ v := \mathbf{cons}(\overline{e})\ \{q\}}$$

$$\Psi_0(\mathcal{A}_0\,v := [e]\,\{q\}) = \Psi_0(\mathcal{A}_0\,\{\exists v''.\ (e \mapsto v'') \ast ((e \mapsto v'') -\!\!\ast\, q'')\}),$$

$$\frac{}{\{\exists v''.\ (e \mapsto v'') \ast ((e \mapsto v'') -\!\!\ast\, q'')\}\ v := [e]\ \{q\}},$$

where, in the last two equations, $q''$ denotes $q/v \to v''$. Then:

**Proposition 14** *If $\mathcal{A} \gg \{p\}\ c\ \{q\}$ is provable by $\mathcal{Q}$, then there is a a proper coherent proof sequence $\mathcal{R}$ from $p$ to $q$ such that:*

1. *If $\Psi_0(\mathcal{A}_0\,\{p\})$ is defined, then:*

   *(a)* $\Psi_0(\mathcal{A}_0\,\mathcal{A})$ *and* $\Psi_0(\mathcal{A}_0\,\mathcal{A}\,\{r\})$ *are defined,*

   *(b)* $\Psi_0(\mathcal{A}_0\,\mathcal{A}) = \Psi_0(\mathcal{A}_0\,\{p\}), \mathcal{R}\,,$

   *(c)* $\Psi_0(\mathcal{A}_0\,\mathcal{A}\,\{r\}) = \Psi_0(\mathcal{A}_0\,\{p\}), \mathcal{R}, q \Rightarrow r\,,$

2. $\Psi_0(\mathcal{A}) = \mathcal{R}\,,$

3. $\Psi(\mathcal{A})$ *is a proof of* $\{p\}\ c\ \{q\}\,,$

4. *The verification conditions in $\Psi(\mathcal{A})$ are the verification conditions in erase-annspec$(\mathcal{Q})$.*

Proof The proof is by induction on the structure of the proof $\mathcal{Q}$. Note that parts 2 and 3 are immediate consequences of Part 1b (taking $\mathcal{A}_0$ to be empty) and the definition of $\Psi$ in terms of $\Psi_0$. END OF PROOF

The developments in this section are summarized by the following diagram, in which $\simeq$ relates proofs containing the same verification conditions, while $\sim$ relates proofs containing the same verification conditions, except perhaps for additional trivial implications of the form $p \Rightarrow p$ in the set of proofs on the right:

$$\begin{tikzcd}
\text{Proofs of Specifications} \arrow[d, "\Phi"] \arrow[dr, dashed, no head, "\sim"] & \\
\text{Proofs of Annotation Descriptions} \arrow[r, "\text{erase-annspec}"] \arrow[d, "\text{concl}"] & \text{Proofs of Specifications} \\
\text{Annotation Descriptions} \arrow[d, "\text{erase-spec}"] & \\
\text{Annotations} \arrow[d, "\Psi"] & \\
\text{Proofs of Specifications} \arrow[uuur, dashed, no head, "\simeq"'] &
\end{tikzcd}$$

# Exercise 1

Fill in the postconditions in

$$\{(e_1 \mapsto -) \ast (e_2 \mapsto -)\}\ [e_1] := e'_1 \,;\, [e_2] := e'_2\ \{?\}$$

$$\{(e_1 \mapsto -) \wedge (e_2 \mapsto -)\}\ [e_1] := e'_1 \,;\, [e_2] := e'_2\ \{?\}.$$

to give two sound inference rules describing a sequence of two mutations. Your postconditions should be as strong as possible.

Give a derivation of each of these inference rules, exhibited as an annotated specification.

# Exercise 2

The alternative inference rule for conditional commands (CDalt), leads to the following rule for annotated specifications:

- Alternative Rule for Conditionals (CDaltan)

$$\frac{\mathcal{A}_1\,\{q\} \gg \{p_1\}\;c_1\;\{q\} \qquad \mathcal{A}_2\,\{q\} \gg \{p_2\}\;c_2\;\{q\}}{\begin{array}{l}(\textbf{if}\ b\ \textbf{then}\ \mathcal{A}_1\ \textbf{else}\ \mathcal{A}_2)\,\{q\} \gg \\ \qquad \{(b \Rightarrow p_1) \wedge (\neg\, b \Rightarrow p_2)\}\ (\textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c_2)\ \{q\},\end{array}}$$

Examine the annotated specifications in this and the following chapters, and determine how they would need to be changed if (CDan) were replaced by (CDaltan).

# Exercise 3

The following are alternative global rules for allocation and lookup that use unmodified variables ($v'$ and $v''$):

- The unmodified-variable global form for allocation (CONSGG)

$$\frac{}{\{v = v' \wedge r\}\ v := \textbf{cons}(\overline{e})\ \{(v \mapsto \overline{e}')\ *\ r'\},}$$

  where $v'$ is distinct from $v$, $\overline{e}'$ denotes $\overline{e}/v \to v'$, and $r'$ denotes $r/v \to v'$.

- The unmodified-variable global form for lookup (LKGG)

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

  where $v$, $v'$, and $v''$ are distinct, $v \notin \mathrm{FV}(r)$, and $e'$ denotes $e/v \to v'$.

Derive (CONSGG) from (CONSG), and (CONSL) from (CONSGG). Derive (LKGG) from (LKG), and (LKL) from (LKGG).

# Exercise 4

Derive (LKNOL) from (LKNOG) and vice-versa. (Hint: To derive (LKNOL) from (LKNOG), use the version of (LKNOG) where $v'' = v$. To derive (LKNOG) from (LKNOL), assume $v$ and $v''$ are distinct, and then apply renaming of $v''$ in the precondition to cover the case where $v = v''$.)

## Exercise 5

Closely akin to (3.1) is the following equivalence of meaning between two lookup commands:

$$v := [e] \cong \textbf{newvar}\ \hat{v}\ \textbf{in}\ (\hat{v} := [e]\, ;\, v := \hat{v}). \tag{3.2}$$

Use this equivalence to derive (LKG) from (LKNOG).

## Exercise 6

Suppose that the *total* correctness specification $[\,p\,]\ c\ [\,q\,]$ holds, and the assertion $q$ is precise. Show, by an informal metaproof, that $p$ is precise.

Hint: Use safety monotonicity, the frame property, and the definition of $[\,p\,]\,c\,[\,q\,]$.