

CLASS NOTES FOR CS 818A3 - SPRING 2005

AN INTRODUCTION TO SEPARATION LOGIC

2. Hoare Logic

John C. Reynolds
Department of Computer Science
Carnegie Mellon University

Revised February 1, 2005

©2005 John C. Reynolds

Expressions

$\langle \text{exp} \rangle ::= 0 \mid 1 \mid 2 \mid \dots$

$\mid \langle \text{var} \rangle \mid - \langle \text{exp} \rangle \mid \langle \text{exp} \rangle + \langle \text{exp} \rangle \mid \langle \text{exp} \rangle - \langle \text{exp} \rangle$

$\mid \langle \text{exp} \rangle \times \langle \text{exp} \rangle \mid \dots$

$\langle \text{boolexp} \rangle ::= \mathbf{true} \mid \mathbf{false}$

$\mid \langle \text{exp} \rangle = \langle \text{exp} \rangle \mid \langle \text{exp} \rangle \neq \langle \text{exp} \rangle \mid \dots$

$\mid \neg \langle \text{boolexp} \rangle \mid \langle \text{boolexp} \rangle \wedge \langle \text{boolexp} \rangle \mid \langle \text{boolexp} \rangle \vee \langle \text{boolexp} \rangle$

$\mid \langle \text{boolexp} \rangle \Rightarrow \langle \text{boolexp} \rangle \mid \langle \text{boolexp} \rangle \Leftrightarrow \langle \text{boolexp} \rangle$

Free Variables

For any phrase p ,

$$\text{FV}(p) = \text{“The variables occurring free in } p\text{”}.$$

There are no binding constructions in expressions or boolean expressions, so that for these phrases $\text{FV}(e)$ is the set of all variables occurring in e .

Stores

$$\text{Stores}_V = V \rightarrow \mathbf{Z},$$

where V is a finite set of variables.

Semantics of Expressions

$$\llbracket e \in \langle \text{exp} \rangle \rrbracket_{\text{exp}} \in \left(\bigcup_{V \supseteq \text{FV}(e)}^{\text{fin}} \text{Stores}_V \right) \rightarrow \mathbf{Z}.$$

$$\llbracket 0 \rrbracket_{\text{exp}} s = 0 \quad (\text{and similarly for } 1, 2, \dots),$$

$$\llbracket v \rrbracket_{\text{exp}} s = sv,$$

$$\llbracket -e \rrbracket_{\text{exp}} s = -\llbracket e \rrbracket_{\text{exp}} s,$$

$$\llbracket e_0 + e_1 \rrbracket_{\text{exp}} s = \llbracket e_0 \rrbracket_{\text{exp}} s + \llbracket e_1 \rrbracket_{\text{exp}} s$$

(and similarly for $-$, \times , \dots).

$$\llbracket b \in \langle \text{boolexp} \rangle \rrbracket_{\text{boolexp}} \in \left(\bigcup_{V \supseteq \text{FV}(b)}^{\text{fin}} \text{Stores}_V \right) \rightarrow \mathbf{B}.$$

$$\llbracket \mathbf{true} \rrbracket_{\text{boolexp}} s = \mathbf{true},$$

$$\llbracket \mathbf{false} \rrbracket_{\text{boolexp}} s = \mathbf{false},$$

$$\llbracket e_0 = e_1 \rrbracket_{\text{boolexp}} s = (\llbracket e_0 \rrbracket_{\text{exp}} s = \llbracket e_1 \rrbracket_{\text{exp}} s)$$

(and similarly for \neq , \dots),

$$\llbracket \neg b \rrbracket_{\text{boolexp}} s = \neg \llbracket b \rrbracket_{\text{boolexp}} s,$$

$$\llbracket b_0 \wedge b_1 \rrbracket_{\text{boolexp}} s = \llbracket b_0 \rrbracket_{\text{boolexp}} s \wedge \llbracket b_1 \rrbracket_{\text{boolexp}} s$$

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

For example:

$$\begin{aligned} \llbracket x + y = x \rrbracket_{\text{boolexp}} s &= (\llbracket x + y \rrbracket_{\text{exp}} s = \llbracket x \rrbracket_{\text{exp}} s) \\ &= (\llbracket x \rrbracket_{\text{exp}} s + \llbracket y \rrbracket_{\text{exp}} s = \llbracket x \rrbracket_{\text{exp}} s) \\ &= (s x + s y = s x) \\ &= (s y = 0). \end{aligned}$$

Substitution

For any phrase p such that $\text{FV}(p) \subseteq \{v_1, \dots, v_n\}$, we write

$$p/v_1 \rightarrow e_1, \dots, v_n \rightarrow e_n$$

to denote the phrase obtained from p by simultaneously substituting each expression e_i for the variable v_i , (When there are bound variables in p , they will be renamed to avoid capture.)

The Substitution Law for Expressions

Proposition 1 *Let δ abbreviate the substitution*

$$v_1 \rightarrow e_1, \dots, v_n \rightarrow e_n,$$

let s be a store such that $\text{FV}(e_1) \cup \dots \cup \text{FV}(e_n) \subseteq \text{dom } s$, and let

$$\widehat{s} = [v_1: \llbracket e_1 \rrbracket_{\text{exp}} s \mid \dots \mid v_n: \llbracket e_n \rrbracket_{\text{exp}} s].$$

If e is an expression (or boolean expression) such that $\text{FV}(e) \subseteq \{v_1, \dots, v_n\}$, then

$$\llbracket e/\delta \rrbracket_{\text{exp}} s = \llbracket e \rrbracket_{\text{exp}} \widehat{s}.$$

Partial Substitution

When $\text{FV}(p)$ is not a subset of $\{v_1, \dots, v_n\}$,

$$p/v_1 \rightarrow e_1, \dots, v_n \rightarrow e_n$$

abbreviates

$$p/v_1 \rightarrow e_1, \dots, v_n \rightarrow e_n, v'_1 \rightarrow v'_1, \dots, v'_k \rightarrow v'_k,$$

where $\{v'_1, \dots, v'_k\} = \text{FV}(p) - \{v_1, \dots, v_n\}$.

Commands

$$\begin{aligned}
 \langle \text{comm} \rangle ::= & \langle \text{var} \rangle := \langle \text{exp} \rangle \mid \mathbf{skip} \mid \langle \text{comm} \rangle ; \langle \text{comm} \rangle \\
 & \mid \mathbf{if} \langle \text{boolexp} \rangle \mathbf{then} \langle \text{comm} \rangle \mathbf{else} \langle \text{comm} \rangle \\
 & \mid \mathbf{while} \langle \text{boolexp} \rangle \mathbf{do} \langle \text{comm} \rangle \\
 & \mid \mathbf{newvar} \langle \text{var} \rangle \mathbf{in} \langle \text{comm} \rangle \\
 & \mid \mathbf{newvar} \langle \text{var} \rangle := \langle \text{exp} \rangle \mathbf{in} \langle \text{comm} \rangle
 \end{aligned}$$

Free Variables of Commands

$$\text{FV}(v := e) = \{v\} \cup \text{FV}(e)$$

$$\text{FV}(\mathbf{skip}) = \{\}$$

$$\text{FV}(c_1 ; c_2) = \text{FV}(c_1) \cup \text{FV}(c_2)$$

$$\text{FV}(\mathbf{if} \ b \ \mathbf{then} \ c_1 \ \mathbf{else} \ c_2) = \text{FV}(b) \cup \text{FV}(c_1) \cup \text{FV}(c_2)$$

$$\text{FV}(\mathbf{while} \ b \ \mathbf{do} \ c) = \text{FV}(b) \cup \text{FV}(c)$$

$$\text{FV}(\mathbf{newvar} \ v \ \mathbf{in} \ c) = \text{FV}(c) - \{v\}$$

$$\text{FV}(\mathbf{newvar} \ v := e \ \mathbf{in} \ c) = \text{FV}(e) \cup (\text{FV}(c) - \{v\}),$$

Compositional Large-Step Semantics of Commands

$$\llbracket c \in \langle \text{comm} \rangle \rrbracket_{\text{comm}} \subseteq \bigcup_{V \stackrel{\text{fin}}{\supseteq} \text{FV}(c)} \text{Stores}_V \times (\text{Stores}_V \cup \{\perp\})$$

is the relation such that

- $s \llbracket c \rrbracket_{\text{comm}} s'$ iff, starting in store s , there is a finite execution of c that terminates in store s' .
- $s \llbracket c \rrbracket_{\text{comm}} \perp$ iff, starting in store s , there is a nonterminating execution of c .

We postpone defining this relation until we consider the more elaborate language used with separation logic. Note, however, that execution preserves the domain of the store, i.e., if $s \llbracket c \rrbracket_{\text{comm}} s'$, then $\text{dom } s = \text{dom } s'$. Also,

Totality

Proposition 2 *Our semantics is total: If $s \in \text{Stores}_V$ for some $V \stackrel{\text{fin}}{\supseteq} \text{FV}(c)$, then there is at least one $\tau' \in \text{Stores}_V \cup \{\perp\}$ such that $s \llbracket c \rrbracket_{\text{comm}} \tau'$.*

Modified Variables

For a command c , the set $\text{MD}(c)$, whose members are called the *variables modified by c* , is defined by structural induction:

$$\text{MD}(v := e) = \{v\}$$

$$\text{MD}(\mathbf{skip}) = \{\}$$

$$\text{MD}(c_0 ; c_1) = \text{MD}(c_0) \cup \text{MD}(c_1)$$

$$\text{MD}(\mathbf{if } b \mathbf{ then } c_0 \mathbf{ else } c_1) = \text{MD}(c_0) \cup \text{MD}(c_1)$$

$$\text{MD}(\mathbf{while } b \mathbf{ do } c) = \text{MD}(c)$$

$$\text{MD}(\mathbf{newvar } v \mathbf{ in } c) = \text{MD}(c) - \{v\}$$

$$\text{MD}(\mathbf{newvar } v := e \mathbf{ in } c) = \text{MD}(c) - \{v\}.$$

Proposition 3 *Suppose $v \in \text{dom}(s)$ and $v \notin \text{MD}(c)$. If*

$$s \llbracket c \rrbracket_{\text{comm}} s',$$

then $s'v = sv$.

Substitution in Commands

Let c be a command such that $\text{FV}(c) \subseteq \{v_1, \dots, v_n\}$, and let δ denote a substitution

$$v_1 \rightarrow e_1, \dots, v_n \rightarrow e_n$$

such that, when $v_i \in \text{MD}(c)$, e_i is a variable. When c is a variable declaration, c/δ is defined by

$$(\mathbf{newvar} \ v \ \mathbf{in} \ c)/\delta = \mathbf{newvar} \ v_{\text{new}} \ \mathbf{in} \ (c/[\delta|v \rightarrow v_{\text{new}}])$$

$$(\mathbf{newvar} \ v := e \ \mathbf{in} \ c)/\delta = \mathbf{newvar} \ v_{\text{new}} := (e/\delta) \ \mathbf{in} \ (c/[\delta|v \rightarrow v_{\text{new}}]),$$

where $v_{\text{new}} \notin \text{FV}(e_1) \cup \dots \cup \text{FV}(e_n)$, and $[\delta|v \rightarrow v_{\text{new}}]$ is the substitution that is like δ except that it maps v into v_{new} .

For other forms of commands,

$$(v_i := e)/\delta = e_i := (e/\delta)$$

$$(\dots p_1 \dots p_n \dots)/\delta = \dots (p_1/\delta) \dots (p_n/\delta) \dots .$$

The Substitution Law for Commands

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

$$v_1 \rightarrow e_1, \dots, v_n \rightarrow e_n$$

such that, when $v_i \in \text{MD}(c)$, e_i is a variable that does not occur in any other e_j .

For any store s such that $\text{FV}(e_1) \cup \dots \cup \text{FV}(e_n) \subseteq \text{dom } s$, let

$$\widehat{s} = [v_1: \llbracket e_1 \rrbracket_{\text{exp}} s \mid \dots \mid v_n: \llbracket e_n \rrbracket_{\text{exp}} s].$$

Then

- *If $s \llbracket c/\delta \rrbracket_{\text{comm}} s'$, then $\widehat{s} \llbracket c \rrbracket_{\text{comm}} \widehat{s}'$.*
- *If $s \llbracket c/\delta \rrbracket_{\text{comm}} \perp$, then $\widehat{s} \llbracket c \rrbracket_{\text{comm}} \perp$.*

Assertions

$$\begin{aligned}
 \langle \text{assert} \rangle &::= \langle \text{boolexp} \rangle \\
 &| \neg \langle \text{assert} \rangle \mid \langle \text{assert} \rangle \wedge \langle \text{assert} \rangle \mid \langle \text{assert} \rangle \vee \langle \text{assert} \rangle \\
 &| \langle \text{assert} \rangle \Rightarrow \langle \text{assert} \rangle \mid \langle \text{assert} \rangle \Leftrightarrow \langle \text{assert} \rangle \\
 &| \forall \langle \text{var} \rangle. \langle \text{assert} \rangle \mid \exists \langle \text{var} \rangle. \langle \text{assert} \rangle
 \end{aligned}$$

In other words, assertions are formulas of the predicate calculus in which the terms are the integer expressions ($\langle \text{exp} \rangle$) of our programming language, and the assertions include the boolean expressions ($\langle \text{boolexp} \rangle$) of the programming language.

Free Variables of Assertions

$$\begin{aligned}
 \text{FV}(\neg p) &= \text{FV}(p) \\
 \text{FV}(p_1 \wedge p_2) &= \text{FV}(p_1) \cup \text{FV}(p_2) \\
 \text{FV}(p_1 \vee p_2) &= \text{FV}(p_1) \cup \text{FV}(p_2) \\
 \text{FV}(p_1 \Rightarrow p_2) &= \text{FV}(p_1) \cup \text{FV}(p_2) \\
 \text{FV}(p_1 \Leftrightarrow p_2) &= \text{FV}(p_1) \cup \text{FV}(p_2) \\
 \text{FV}(\forall v. p) &= \text{FV}(p) - \{v\} \\
 \text{FV}(\exists v. p) &= \text{FV}(p) - \{v\}
 \end{aligned}$$

The Semantics of Assertions

$$\llbracket p \in \langle \text{assert} \rangle \rrbracket_{\text{assert}} \in \left(\bigcup_{V \supseteq \text{FV}(p)}^{\text{fin}} \text{Stores}_V \right) \rightarrow \mathbf{B}.$$

Instead of $\llbracket p \rrbracket_{\text{assert}} s = \mathbf{true}$, however, we write $s \models p$.

$$s \models b \text{ iff } \llbracket b \rrbracket_{\text{boolexp}} s = \mathbf{true},$$

$$s \models \neg p \text{ iff } s \models p \text{ is false,}$$

$$s \models p_0 \wedge p_1 \text{ iff } s \models p_0 \text{ and } s \models p_1$$

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

$$s \models \forall v. p \text{ iff } \forall x \in \mathbf{Z}. [s \mid v: x] \models p,$$

$$s \models \exists v. p \text{ iff } \exists x \in \mathbf{Z}. [s \mid v: x] \models p,$$

where $[s \mid x: n] y = \mathbf{if } x = y \mathbf{ then } n \mathbf{ else } s(y)$.

Examples

$$s \models x + y = x \text{ iff } \llbracket x + y = x \rrbracket_{\text{boolexp}} s$$

$$\text{iff } s y = 0.$$

$$s \models \forall x. x + y = x \text{ iff } \forall n \in \mathbf{Z}. [s \mid x: n] \models x + y = x$$

$$\text{iff } \forall n \in \mathbf{Z}. [s \mid x: n] y = 0$$

$$\text{iff } \forall n \in \mathbf{Z}. s y = 0$$

$$\text{iff } s y = 0.$$

Substitution in Assertions

Let p be an assertion such that $\text{FV}(p) \subseteq \{v_1, \dots, v_n\}$, and let δ denote a substitution

$$v_1 \rightarrow e_1, \dots, v_n \rightarrow e_n.$$

When p begins with a quantifier, p/δ is defined by

$$\begin{aligned} (\forall v. p)/\delta &= \forall v_{\text{new}}. (p/[\delta|v \rightarrow v_{\text{new}}]) \\ (\exists v. p)/\delta &= \exists v_{\text{new}}. (p/[\delta|v \rightarrow v_{\text{new}}]) \end{aligned}$$

where $v_{\text{new}} \notin \text{FV}(e_1) \cup \dots \cup \text{FV}(e_n)$, and $[\delta|v \rightarrow v_{\text{new}}]$ is the substitution that is like δ except that it maps v into v_{new} .

For other forms of assertions:

$$(\dots p_1 \dots p_n \dots)/\delta = \dots (p_1/\delta) \dots (p_n/\delta) \dots$$

The Substitution Law for Assertions

Proposition 5 *Let δ abbreviate the substitution*

$$v_1 \rightarrow e_1, \dots, v_n \rightarrow e_n,$$

let s be a store such that $\text{FV}(e_1) \cup \dots \cup \text{FV}(e_n) \subseteq \text{dom } s$, and let

$$\widehat{s} = [v_1: \llbracket e_1 \rrbracket_{\text{exp}} s \mid \dots \mid v_n: \llbracket e_n \rrbracket_{\text{exp}} s].$$

If p is an assertion such that $\text{FV}(p) \subseteq \{v_1, \dots, v_n\}$, then

$$s \vdash (p/\delta) \text{ iff } \widehat{s} \vdash p.$$

Some Cautions

- It is vital to distinguish between *variables* (in programs or assertions) and *metavariables* (which range over phrases of the programming language or logic). We will use san-serif font for variables and Roman or Greek fonts for metavariables.
- We suppress any indication of a model or structure, since, with rare exceptions, the only model we will use for the predicate calculus is the standard model \mathcal{M}_{int} of the integers. In place of $\mathcal{M}_{\text{int}} \models_s p$ we write $s \models p$, and we call s a *store*, rather than an *assignment* or *environment*.
- In standard Hoare logic (and in separation logic), expressions in the programming language are the same as terms in the predicate calculus, and have the same meaning. They always terminate, never give error stops, and never have side effects.

Some Terminology

Let V contain all of the free variables of p (or p and q in cases 4 and 5). Then

1. When $s \models p$ for some store $s \in \text{Store}_V$, we say
 - p is *true* in s ,
 - p *holds* for s ,
 - p *describes* s ,
 - s *satisfies* p .
2. When $s \models p$ for all $s \in \text{Store}_V$, we say
 - p is *valid*.
3. When $s \models p$ for no $s \in \text{Store}_V$, we say
 - p is *unsatisfiable*.
4. When $s \models q$ holds for every $s \in \text{Store}_V$ such that $s \models p$, we say
 - p is *stronger* than q ,
 - q is *weaker* than p .

(Notice that **true** is weaker than any assertion and **false** is stronger than any assertion.)

5. When $s \models q$ holds for exactly those $s \in \text{Store}_V$ such that $s \models p$, we say
 - p is *equivalent* to q .

Specifications (Hoare Triples)

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

$| [\langle \text{assert} \rangle] \langle \text{comm} \rangle [\langle \text{assert} \rangle]$ (total correctness)

Let $V = \text{FV}(p) \cup \text{FV}(c) \cup \text{FV}(q)$. Then

Partial correctness:

$\{p\} c \{q\}$ holds iff $\forall s \in \text{Stores}_V. s \models p$ implies
 $(\forall s' \in \text{Stores}_V. s \llbracket c \rrbracket_{\text{comm}} s' \text{ implies } s' \models q)$.

Total correctness:

$[p] c [q]$ holds iff $\forall s \in \text{Stores}_V. s \models p$ implies
 $\neg s \llbracket c \rrbracket_{\text{comm}} \perp$
 and $(\forall s' \in \text{Stores}_V. s \llbracket c \rrbracket_{\text{comm}} s' \text{ implies } s' \models q)$.

Examples of Valid Partial Correctness Specifications

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

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

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

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

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

Examples of Valid Total Correctness Specifications

$$[x - y > 3] x := x - y [x > 3]$$

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

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

Inference Rules and Proofs

- An *inference rule* for Hoare logic consists of zero or more *premisses* (either specifications or assertions) and a single *conclusion* (a specification), separated by a horizontal line:

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

- The premisses and conclusion are schemata, i.e., they may contain *metavariables*, each of which ranges over some set of phrases, such as expressions, commands, or assertions.
- An instance of an inference rule is obtained by replacing each metavariable by a phrase in its range. These replacements must satisfy the *side-conditions* (if any) of the rule. (Since this is replacement of metavariables rather than substitution for variables, there is never any renaming.)
- A *formal proof* in Hoare logic is a sequence of assertions and/or specifications, each of which is either a valid assertion or the conclusion of some instance of a sound inference rule whose premisses occur earlier in the sequence,
- An inference rule with zero premisses is called an *axiom schema*. The overbar is sometimes omitted.
- An axiom schema containing no metavariables is called an *axiom*. The overbar is usually omitted.

Soundness

An inference rule of Hoare logic is *sound* iff, for all instances,

- if the premisses of the instance are all valid,
- then the conclusion is valid.

Since we require the assertions in a formal proof to be valid and the inference rules used in the proof to be sound, it follows that the specifications in a formal proof must all be valid.

Inference Rules for Specifications:

Assignment (AS)

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

Instances

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

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

Inference Rules for Specifications:

Sequential Composition (SQ)

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

An Instance

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

Note

Except for **while** commands and recursive procedures, reasoning for partial and total correctness is the same. Henceforth, we will give only rules for partial correctness, except when the rule for total correctness is different.

Inference Rules for Specifications:

Strengthening Precedent (SP)

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

An Instance

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

Weakening Consequent (WC)

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

An Instance

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

Since they are applicable to arbitrary commands, the rules (SP) and (WC) are called *structural rules*. One premiss of each of these rules is an assertion, which is called a *verification condition* (VC). The verification conditions are used to introduce mathematical facts about data types into proofs of specifications.

A Simple Proof

$$1. \quad \{(2 \times y) - y > 3\} x := 2 \times y \{x - y > 3\} \quad (\text{AS})$$

$$2. \quad \{x - y > 3\} x := x - y \{x > 3\} \quad (\text{AS})$$

$$3. \quad \{(2 \times y) - y > 3\} x := 2 \times y ; x := x - y \{x > 3\} \quad (\text{SQ},1,2)$$

$$4. \quad y > 3 \Rightarrow (2 \times y) - y > 3 \quad (\text{VC})$$

$$5. \quad \{y > 3\} x := 2 \times y ; x := x - y \{x > 3\} \quad (\text{SP},4,3)$$

$$6. \quad x > 3 \Rightarrow x \geq 4 \quad (\text{VC})$$

$$7. \quad \{y > 3\} x := 2 \times y ; x := x - y \{x \geq 4\} \quad (\text{WC},5,6)$$

As long as only simple facts about integer arithmetic are involved, we will allow valid verification conditions to appear in proofs without proving these conditions.

A More Realistic Proof

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

A More Realistic Proof (continued)

6. $\{f + g = \text{fib}(k + 1) \wedge f = \text{fib}((k + 1) - 1) \wedge k + 1 \leq n\}$
 $t := g$
 $\{f + t = \text{fib}(k + 1) \wedge f = \text{fib}((k + 1) - 1) \wedge k + 1 \leq n\}$ (AS)
7. $\{f + g = \text{fib}(k + 1) \wedge f = \text{fib}((k + 1) - 1) \wedge k + 1 \leq n\}$
 $t := g ; g := f ; f := f + t ; k := k + 1$
 $\{f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n\}$ (SQ,5,6)
8. $(f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n \wedge k \neq n) \Rightarrow$
 $(f + g = \text{fib}(k + 1) \wedge f = \text{fib}((k + 1) - 1) \wedge k + 1 \leq n)$
(VC*)
9. $\{f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n \wedge k \neq n\}$
 $t := g ; g := f ; f := f + t ; k := k + 1$
 $\{f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n\}$ (SP,7,8)

*since

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

Inference Rules for Specifications:

An Alternative “Forward” Rule for Assignment (ASALT)

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

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

The problem with this rule is the accumulation of quantifiers. For example, when we use it to prove

$$\begin{aligned} & \{f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n \wedge k \neq n\} \\ & t := g ; g := f ; f := f + t ; k := k + 1 \\ & \{f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n\}, \end{aligned}$$

we get the verification condition

$$\begin{aligned} & (\exists k'. \exists f'. \exists g'. k = k' + 1 \wedge f = f' + t \wedge g = f' \wedge t = g' \\ & \quad \wedge f' = \text{fib}(k') \wedge g' = \text{fib}(k' - 1) \wedge k' \leq n \wedge k' \neq n) \Rightarrow \\ & f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n. \end{aligned}$$

Inference Rules for Specifications:

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 .

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

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

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

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

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

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

An Instance of the Rule for Declarations

$$\begin{array}{l}
 \{f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n \wedge k \neq n\} \\
 t := g ; g := f ; f := f + t ; k := k + 1 \\
 \{f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n\} \\
 \hline
 \{f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n \wedge k \neq n\} \\
 \mathbf{newvar\ } t \mathbf{ in\ } (t := g ; g := f ; f := f + t ; k := k + 1) \\
 \{f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n\}
 \end{array}$$

Inference Rules for Specifications:

Partial Correctness of **while** (WHP)

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

Here i is the *invariant*.

An Instance

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

Inference Rules for Specifications:

Total Correctness of **while** (WHT)

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

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

Here i is the *invariant*, e is the *variant*, and v_0 is a *ghost variable*.

An Instance

$$[f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n \wedge k \neq n \wedge n - k = v_0]$$

newvar t in (t := g ; g := f ; f := f + t ; k := k + 1)

$$[f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n \wedge n - k < v_0]$$

$$(f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n \wedge k \neq n) \Rightarrow n - k \geq 0$$

$$[f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n]$$

while $k \neq n$ **do**

newvar t in (t := g ; g := f ; f := f + t ; k := k + 1)

$$[f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n \wedge \neg k \neq n]$$

Other Well-Ordered Sets

Sometimes, the range of variants must be a larger well-ordered set than the set of natural numbers. For example, to show the termination of

```
[true]
while x ≠ 0 do
  if x < 0 then (newvar y in x := |y|) else x := x - 1
[x = 0]
```

we can use the variant **if** $x < 0$ **then** ∞ **else** x , which ranges over the well-ordered set $\{n \mid n \geq 0\} \cup \{\infty\}$.

This program contains the command

$$\mathbf{newvar\ } y \mathbf{\ in\ } x := |y|,$$

which always terminates, yet has an infinite number of possible outcomes. This is called *unbounded nondeterminism*.

Inference Rules for Specifications:

Conditional (CD)

$$\frac{\{p \wedge b\} c_1 \{q\} \quad \{p \wedge \neg b\} c_2 \{q\}}{\{p\} \text{ if } b \text{ then } c_1 \text{ else } c_2 \{q\}}$$

An Instance

$\{n \geq 0 \wedge n = 0\}$

$f := 0$

$\{f = \text{fib}(n)\}$

$\{n \geq 0 \wedge \neg n = 0\}$

newvar g in newvar k in

$(k := 1 ; g := 0 ; f := 1 ;$

while $k \neq n$ **do**

newvar t in $(t := g ; g := f ; f := f + t ; k := k + 1)$

$\{f = \text{fib}(n)\}$

$\{n \geq 0\}$

if $n = 0$ **then** $f := 0$ **else**

newvar g in newvar k in

$(k := 1 ; g := 0 ; f := 1 ;$

while $k \neq n$ **do**

newvar t in $(t := g ; g := f ; f := f + t ; k := k + 1)$

$\{f = \text{fib}(n)\}$

Inference Rules for Specifications:

An Alternative Rule for Conditionals (CDALT)

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

skip (SK)

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

Annotated Specifications

In an *annotated specification*, additional assertions called *annotations* are placed within the command in such a way that the proof of the specification can be reconstructed from the annotations. For example, the following is a (highly) annotated total correctness specification for the Fibonacci program:

```

[n ≥ 0]
if n = 0 then
    [0 = fib(n)]
    f := 0
else
    newvar g in newvar k in
        ([1 = fib(1) ∧ 0 = fib(1 - 1) ∧ 1 ≤ n]
        k := 1 ;
        [1 = fib(k) ∧ 0 = fib(k - 1) ∧ k ≤ n]
        g := 0 ;
        [1 = fib(k) ∧ g = fib(k - 1) ∧ k ≤ n]
        f := 1 ;
        [f = fib(k) ∧ g = fib(k - 1) ∧ k ≤ n]
        while [vrnt: n - k] k ≠ n do newvar t in (⋯)
        [f = fib(k) ∧ g = fib(k - 1) ∧ k ≤ n ∧ ¬k ≠ n])
    [f = fib(n)]

```

Annotated Specifications (continued)

 $[n \geq 0]$
 \vdots
while $[vrnt: n - k] k \neq n$ **do newvar** t **in**
 $([f+g = fib(k+1) \wedge f = fib((k+1)-1) \wedge k+1 \leq n \wedge n-(k+1) < v_0]$
 $t := g;$
 $[f+t = fib(k+1) \wedge f = fib((k+1)-1) \wedge k+1 \leq n \wedge n-(k+1) < v_0]$
 $g := f;$
 $[f+t = fib(k+1) \wedge g = fib((k+1)-1) \wedge k+1 \leq n \wedge n-(k+1) < v_0]$
 $f := f + t;$
 $[f = fib(k+1) \wedge g = fib((k+1)-1) \wedge k+1 \leq n \wedge n-(k+1) < v_0]$
 $k := k + 1)$
 $([f = fib(k) \wedge g = fib(k - 1) \wedge k \leq n \wedge \neg k \neq n])$
 $[f = fib(n)]$

Why Annotations Are Needed

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,

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

to obtain the main step of a proof of the specification

```
{n ≥ 0}
(k := 0 ; s := 1) ;
while k ≠ n do (k := k + 1 ; s := 2 × s)
{s = 2n},
```

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

Why Annotations Are Needed (continued)

But if we change the rule to

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

then the new rule requires the annotation q to occur in the conclusion:

```
{n ≥ 0}
(k := 0 ; s := 1) ;
{s = 2k ∧ k ≤ n}
while k ≠ n do (k := k + 1 ; s := 2 × s)
{s = 2n}.
```

Then, once q is determined, the premisses must be

```
{n ≥ 0}
(k := 0 ; s := 1);
{s = 2k ∧ k ≤ n}
```

and

```
{s = 2k ∧ k ≤ n}
while k ≠ n do (k := k + 1 ; s := 2 × s)
{s = 2n}.
```

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.

Rules for Annotated Specifications

To obtain annotated specifications, we must change the following rules (and avoid using the “alternative” rules given earlier):

Sequential Composition (SQAN)

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

Strengthening Precedent (SPAN)

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

Weakening Consequent (WCAN)

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

Total Correctness of **while**

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

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

More Rules for Annotated Specifications

In their present form, our inference rules lead to many more annotations than necessary. To reduce the number of annotations, it is useful to change further rules. (Each of the following rules can be derived from earlier rules.)

Assignment (ASAN)

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

skip (SKAN)

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

Partial Correctness of **while** (WHPAN)

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

Total Correctness of **while** (WHTAN)

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

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

A Minimally Annotated Specification for Fibonacci Numbers

```

[n ≥ 0]
if n = 0 then f := 0 else
  newvar g in newvar k in
    (k := 1 ; g := 0 ; f := 1 ;
     [f = fib(k) ∧ g = fib(k - 1) ∧ k ≤ n]
     while [vrnt: n - k] k ≠ n do
       newvar t in (t := g ; g := f ; f := f + t ; k := k + 1))
[f = fib(n)]

```

Verification Conditions

1. $n \geq 0 \wedge n = 0 \Rightarrow 0 = \text{fib}(n)$
2. $n \geq 0 \wedge \neg n = 0 \Rightarrow 1 = \text{fib}(1) \wedge 0 = \text{fib}(1 - 1) \wedge 1 \leq n$
3. $f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n \wedge k \neq n \wedge n - k = v_0 \Rightarrow$
 $f + g = \text{fib}(k + 1) \wedge f = \text{fib}((k + 1) - 1) \wedge k + 1 \leq n$
 $\wedge n - (k + 1) < v_0$
4. $f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n \wedge k \neq n \Rightarrow n - k \geq 0$
5. $f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n \wedge \neg k \neq n \Rightarrow f = \text{fib}(n)$

The Proof Determined by the Annotated Specification

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

$$2. \quad [n \geq 0 \wedge n = 0]$$

$$\mathbf{f} := 0$$

$$[\mathbf{f} = \text{fib}(n)] \quad (\text{ASAN1,1})$$

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

$$4. \quad [n \geq 0 \wedge \neg n = 0]$$

$$\mathbf{k} := 1$$

$$[1 = \text{fib}(k) \wedge 0 = \text{fib}(k - 1) \wedge k \leq n] \quad (\text{ASAN1,3})$$

$$5. \quad [n \geq 0 \wedge \neg n = 0]$$

$$\mathbf{k} := 1 ; \mathbf{g} := 0$$

$$[1 = \text{fib}(k) \wedge \mathbf{g} = \text{fib}(k - 1) \wedge k \leq n] \quad (\text{ASAN2,4})$$

$$6. \quad [n \geq 0 \wedge \neg n = 0]$$

$$\mathbf{k} := 1 ; \mathbf{g} := 0 ; \mathbf{f} := 1$$

$$[\mathbf{f} = \text{fib}(k) \wedge \mathbf{g} = \text{fib}(k - 1) \wedge k \leq n] \quad (\text{ASAN2,5})$$

$$7. \quad (\mathbf{f} = \text{fib}(k) \wedge \mathbf{g} = \text{fib}(k - 1) \wedge k \leq n \wedge k \neq n \wedge n - k = v_0) \Rightarrow \\ (\mathbf{f} + \mathbf{g} = \text{fib}(k + 1) \wedge \mathbf{f} = \text{fib}((k + 1) - 1) \wedge k + 1 \leq n \wedge \\ n - (k + 1) < v_0) \quad (\text{VC})$$

8. $[f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n \wedge k \neq n \wedge n - k = v_0]$
 $t := g$
 $[f + t = \text{fib}(k + 1) \wedge f = \text{fib}((k + 1) - 1) \wedge k + 1 \leq n \wedge$
 $n - (k + 1) < v_0]$ (ASAN1,7)
9. $[f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n \wedge k \neq n \wedge n - k = v_0]$
 $t := g ; g := f$
 $[f + t = \text{fib}(k + 1) \wedge g = \text{fib}((k + 1) - 1) \wedge k + 1 \leq n \wedge$
 $n - (k + 1) < v_0]$ (ASAN2,8)
10. $[f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n \wedge k \neq n \wedge n - k = v_0]$
 $t := g ; g := f ; f := f + t$
 $[f = \text{fib}(k + 1) \wedge g = \text{fib}((k + 1) - 1) \wedge k + 1 \leq n \wedge$
 $n - (k + 1) < v_0]$ (ASAN2,9)
11. $[f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n \wedge k \neq n \wedge n - k = v_0]$
 $t := g ; g := f ; f := f + t ; k := k + 1$
 $[f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n \wedge n - k < v_0]$
 (ASAN2,10)
12. $[f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n \wedge k \neq n \wedge n - k = v_0]$
 $\mathbf{newvar} \ t \ \mathbf{in} \ (t := g ; g := f ; f := f + t ; k := k + 1)$
 $[f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n \wedge n - k < v_0]$ (DC,11)

$$13. (f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n \wedge k \neq n) \Rightarrow n - k \geq 0$$

(VC)

$$14. (f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n \wedge \neg k \neq n) \Rightarrow f = \text{fib}(n)$$

(VC)

$$15. [f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n]$$

while [vrnt: $n - k$] $k \neq n$ **do**

newvar t **in** ($t := g ; g := f ; f := f + t ; k := k + 1$)

[$f = \text{fib}(n)$] (WHTAN,12,13,14)

$$16. [n \geq 0 \wedge \neg n = 0]$$

$k := 1 ; g := 0 ; f := 1 ;$

[$f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n$]

while [vrnt: $n - k$] $k \neq n$ **do**

newvar t **in** ($t := g ; g := f ; f := f + t ; k := k + 1$)

[$f = \text{fib}(n)$] (SQAN,6,15)

$$17. [n \geq 0 \wedge \neg n = 0]$$

newvar k **in**

($k := 1 ; g := 0 ; f := 1 ;$

[$f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n$]

while [vrnt: $n - k$] $k \neq n$ **do**

newvar t **in** ($t := g ; g := f ; f := f + t ; k := k + 1$))

[$f = \text{fib}(n)$] (DC,16)

18. $[n \geq 0 \wedge \neg n = 0]$

newvar g in newvar k in

$(k := 1 ; g := 0 ; f := 1 ;$

$[f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n]$

while [vrnt: $n - k$] $k \neq n$ do

newvar t in $(t := g ; g := f ; f := f + t ; k := k + 1)$

$[f = \text{fib}(n)]$

(DC,17)

19. $[n \geq 0]$

if $n = 0$ then $f := 0$ else

newvar g in newvar k in

$(k := 1 ; g := 0 ; f := 1 ;$

$[f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n]$

while [vrnt: $n - k$] $k \neq n$ do

newvar t in $(t := g ; g := f ; f := f + t ; k := k + 1)$

$[f = \text{fib}(n)]$

(COND,2,18)

An Annotated Specification for Fast Exponentiation

$[n \geq 0]$

newvar k in newvar z in

$(k := n ; z := x ; y := 1 ;$

$[y \times z^k = x^n \wedge k \geq 0]$

while [vrnt: k] $k \neq 0$ do

(if odd(k) then $(k := k - 1 ; y := y \times z)$ else skip ;

$k := k \div 2 ; z := z \times z)$)

$[y = x^n]$

Verification Conditions

1. $n \geq 0 \Rightarrow 1 \times x^n = x^n \wedge n \geq 0$
2. $y \times z^k = x^n \wedge k \geq 0 \wedge k \neq 0 \wedge k = k_0 \wedge \mathbf{odd}(k) \Rightarrow$
 $(y \times z) \times (z \times z)^{(k-1) \div 2} = x^n$
 $\wedge (k-1) \div 2 \geq 0 \wedge (k-1) \div 2 < k_0$
3. $y \times z^k = x^n \wedge k \geq 0 \wedge k \neq 0 \wedge k = k_0 \wedge \neg \mathbf{odd}(k) \Rightarrow$
 $y \times (z \times z)^{k \div 2} = x^n \wedge k \div 2 \geq 0 \wedge k \div 2 < k_0$
4. $y \times z^k = x^n \wedge k \geq 0 \wedge k \neq 0 \Rightarrow k \geq 0$
5. $y \times z^k = x^n \wedge k \geq 0 \wedge \neg k \neq 0 \Rightarrow y = x^n$

Interderivability of the Assignment Rules

$$\overline{\{p/v \rightarrow e\} v := e \{p\}}$$

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

where $v' \notin \{v\} \cup \text{FV}(e) \cup \text{FV}(p)$, e' is $e/v \rightarrow v'$, and p' is $p/v \rightarrow v'$.

To show the second rule, assuming the first:

$$\begin{array}{l} \{p\} \\ \{e = e \wedge p\} \\ \{(e = e' \wedge p')/v' \rightarrow v\} \\ \{\exists v'. e = e' \wedge p'\} \\ v := e \\ \{\exists v'. v = e' \wedge p'\} \end{array}$$

To show the first rule, assuming the second:

$$\begin{array}{l} \{p/v \rightarrow e\} \\ v := e \\ \{\exists v'. v = e' \wedge ((p/v \rightarrow e)/v \rightarrow v')\} \\ \{\exists v'. v = e' \wedge (p/v \rightarrow e')\} \\ \{\exists v'. v = e' \wedge p\} \\ \{\exists v'. p\} \\ \{p\} \end{array}$$

Note that these are *proof schemata*, i.e., they become proofs when the metavariables are replaced by appropriate phrases.

Interderivability of the Conditional Rules

$$\frac{\frac{\{p \wedge b\} c_1 \{q\} \quad \{p \wedge \neg b\} c_2 \{q\}}{\{p\} \text{ if } b \text{ then } c_1 \text{ else } c_2 \{q\}}}{\frac{\{p_1\} c_1 \{q\} \quad \{p_2\} c_2 \{q\}}{\{(b \Rightarrow p_1) \wedge (\neg b \Rightarrow p_2)\} \text{ if } b \text{ then } c_1 \text{ else } c_2 \{q\}}}$$

Assume the premises of the second rule. Then

$$\begin{array}{l}
 \{(b \Rightarrow p_1) \wedge (\neg b \Rightarrow p_2) \wedge b\} \\
 \{p_1\} \\
 c_1 \\
 \{q\}
 \end{array}$$

and

$$\begin{array}{l}
 \{(b \Rightarrow p_1) \wedge (\neg b \Rightarrow p_2) \wedge \neg b\} \\
 \{p_2\} \\
 c_2 \\
 \{q\},
 \end{array}$$

and the first rule gives the conclusion of the second rule:

$$\begin{array}{l}
 \{(b \Rightarrow p_1) \wedge (\neg b \Rightarrow p_2)\} \\
 \text{if } b \text{ then } c_1 \text{ else } c_2 \\
 \{q\}.
 \end{array}$$

Interderivability of the Conditional Rules (continued)

$$\frac{\frac{\{p \wedge b\} c_1 \{q\} \quad \{p \wedge \neg b\} c_2 \{q\}}{\{p\} \text{ if } b \text{ then } c_1 \text{ else } c_2 \{q\}}}{\frac{\{p_1\} c_1 \{q\} \quad \{p_2\} c_2 \{q\}}{\{(b \Rightarrow p_1) \wedge (\neg b \Rightarrow p_2)\} \text{ if } b \text{ then } c_1 \text{ else } c_2 \{q\}}}$$

Assume the premises of the first rule. Then the second rule gives the main step in

$$\begin{array}{l}
 \{p\} \\
 \{(b \Rightarrow p \wedge b) \wedge (\neg b \Rightarrow p \wedge \neg b)\} \\
 \text{if } b \text{ then } c_1 \text{ else } c_2 \\
 \{q\},
 \end{array}$$

which is the conclusion of the second rule.

Structural Inference Rules

In addition to Strengthening Precedent and Weakening Consequent, there are a number of other *structural* rules that are applicable to all commands.

Renaming (RN)

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

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

It is sometimes necessary to use renaming in combination with the rule for variable declarations, e.g.

$$\frac{\{x = 0\} y := 1 \{x = 0\}}{\{x = 0\} \mathbf{newvar} y \mathbf{in} y := 1 \{x = 0\}}$$

$$\{x = 0\} \mathbf{newvar} x \mathbf{in} x := 1 \{x = 0\}.$$

Substitution (SUB)

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

where v_1, \dots, 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 .

For example, in

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

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

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

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

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

The rule for substitution will become important when we consider procedures.

A special case of this rule occurs for an *identity* substitution, in which each e_i is v_i . Such substitutions still permit renaming (by choosing the v_{new}). Thus the Renaming rule is a special case of the substitution rule.

Structural Inference Rules:

Conjunction (CONJ)

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

Disjunction (DISJ)

$$\frac{\{p_1\} c \{q_1\} \quad \{p_2\} c \{q_2\}}{\{p_1 \vee p_2\} c \{q_1 \vee q_2\}}$$

Universal Quantification (UQ)

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

where v is not free in c .

Existential Quantification (EQ)

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

where v is not free in c .

Structural Inference Rules:

Constancy (CONST)

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

when no variable occurring free in r is modified by c .

Structural Rules for Annotated Specifications

The use of the structural rules for renaming, substitution, disjunction, and conjunction cannot easily be indicated in annotated specifications. But we can give versions of the remaining rules that are suitable:

Existential Quantification (EQAN)

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

where v is not free in c .

Universal Quantification (UQAN)

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

where v is not free in c .

Constancy (CONSTAN)

$$\frac{\{p\} c \{q\}}{\{p \wedge r\} \mathbf{coninv} r \mathbf{in} c \{q \wedge r\}}$$

when no variable occurring free in r is modified by c .

Ghost Variables

A *ghost variable* of a specification is a variable that occurs in the specification but not in the command being specified.

The main use of ghost variables is to relate stores at different points in the program. For example, in the following program, the ghost variable k_0 is used to “remember” the initial value of k , in order to specify that the final value of y is x raised to the initial value of k :

$[k \geq 0 \wedge k = k_0]$

newvar z **in**

$(z := x ; y := 1 ;$

$[y \times z^k = x^{k_0} \wedge k \geq 0]$

while $[vrnt: k] k \neq 0$ **do**

$(\text{if odd}(k) \text{ then } (k := k - 1 ; y := y \times z) \text{ else skip ;}$

$k := k \div 2 ; z := z \times z)$

$[y = x^{k_0}]$

Totality is built into the Predicate Calculus

Suppose we permit an ill-defined expression such as $x \div 0$. Then our rules of inference allow us to prove such specifications as:

$$[\mathbf{true}] y := x \div 0 ; y := 7 [y = 7]$$

$$[x = 7] \mathbf{if} x \div 0 = 29 \mathbf{then} x := x + 1 \mathbf{else} x := x - 1 [x = 8 \vee x = 6]$$

$$[x = 7] \mathbf{if} x \div 0 = x \div 0 \mathbf{then} x := x + 1 \mathbf{else} x := x - 1 [x = 8].$$

Even in the absence of any axioms about \div , the assumption is built into our logic that $x \div 0$ terminates without an error stop or side effects, and always gives the same value.

Varied Arithmetic

In general, integer division satisfies

$$\begin{aligned} y \neq 0 &\Rightarrow x = (x \div y) \times y + x \mathbf{rem} y \\ x \geq 0 \wedge y > 0 &\Rightarrow 0 \leq x \mathbf{rem} y < y, \end{aligned} \tag{1}$$

However, although most machines provide division that is “odd” in x and y :

$$\begin{aligned} y \neq 0 &\Rightarrow (-x) \div y = -(x \div y) \\ y \neq 0 &\Rightarrow x \div (-y) = -(x \div y), \end{aligned}$$

a few machines provide a “number-theoretic” division satisfying

$$y > 0 \Rightarrow 0 \leq x \mathbf{rem} y < y,$$

so that, for example, $-3 \div 2 = -2$ and $-3 \mathbf{rem} 2 = 1$.

If one limits the axioms about division to (1), then anything one proves will hold for both kinds of arithmetic. One can also add a monotonicity law, since it holds in both situations:

$$y > 0 \wedge x \leq x' \Rightarrow x \div y \leq x' \div y.$$

Weakest Preconditions

We extend our language of assertions with notations for *weakest preconditions* and *weakest liberal preconditions*:

$$\begin{aligned} \langle \text{assert} \rangle ::= & \dots \\ & | \mathbf{wp}(\langle \text{comm} \rangle, \langle \text{assert} \rangle) \\ & | \mathbf{wlp}(\langle \text{comm} \rangle, \langle \text{assert} \rangle) \end{aligned}$$

with the semantics

$$\begin{aligned} s \models \mathbf{wlp}(c, q) \text{ iff } & \forall s' \in \text{Stores}_V. s \llbracket c \rrbracket_{\text{comm}} s' \text{ implies } s' \models q \\ s \models \mathbf{wp}(c, q) \text{ iff } & \neg s \llbracket c \rrbracket_{\text{comm}} \perp \text{ and} \\ & (\forall s' \in \text{Stores}_V. s \llbracket c \rrbracket_{\text{comm}} s' \text{ implies } s' \models q). \end{aligned}$$

It follows that specifications can be defined as:

$$\begin{aligned} \{p\} c \{q\} \text{ iff } & p \Rightarrow \mathbf{wlp}(c, q) \text{ is valid,} \\ [p] c [q] \text{ iff } & p \Rightarrow \mathbf{wp}(c, q) \text{ is valid.} \end{aligned}$$

General Rules

The following inference rules for our extended assertion language are sound, in the sense that, when all the premisses of an instance are valid, then the conclusion is valid.

- Law of the Excluded Miracle

$$\frac{}{\mathbf{wp}(c, \mathbf{false}) \Leftrightarrow \mathbf{false}},$$

- Monotonicity

$$\frac{p \Rightarrow q}{\mathbf{wp}(c, p) \Rightarrow \mathbf{wp}(c, q)},$$

- Conjunction

$$\frac{}{(\mathbf{wp}(c, p) \wedge \mathbf{wp}(c, q)) \Leftrightarrow \mathbf{wp}(c, p \wedge q)},$$

- Disjunction

$$\frac{}{(\mathbf{wp}(c, p) \vee \mathbf{wp}(c, q)) \Rightarrow \mathbf{wp}(c, p \vee q)}.$$

Note that the inference rules (CONJ) and (DISJ) follow from the above rules for conjunction and disjunction.

The analogous rules hold for **wlp**, except for the law of the excluded miracle.

Rules for Specific Commands

For most commands, one can give an axiom schema that expresses the weakest precondition in terms of the weakest preconditions of subcommands:

- Assignment

$$\frac{}{\mathbf{wp}(v := e, q) \Leftrightarrow (q/v \rightarrow e)},$$

- skip

$$\frac{}{\mathbf{wp}(\mathbf{skip}, q) \Leftrightarrow q},$$

- Sequential Composition

$$\frac{}{\mathbf{wp}(c_0 ; c_1, q) \Leftrightarrow \mathbf{wp}(c_0, \mathbf{wp}(c_1, q))},$$

- Conditional

$$\frac{}{\mathbf{wp}(\mathbf{if } b \mathbf{ then } c_0 \mathbf{ else } c_1, q) \Leftrightarrow (b \Rightarrow \mathbf{wp}(c_0, q)) \wedge (\neg b \Rightarrow \mathbf{wp}(c_1, q))}.$$

Note that the inference rules (AS), (SK), (SQ), and (CDALT) follow from the above rules.

Rules for Specific Commands (continued)

- Uninitialized Variable Declaration

$$\frac{}{\mathbf{wp}(\mathbf{newvar} \ v \ \mathbf{in} \ c, q) \Leftrightarrow (\forall v. \ \mathbf{wp}(c, q))}$$

when v is not free in q ,

- Initialized Variable Declaration

$$\frac{}{\mathbf{wp}(\mathbf{newvar} \ v := e \ \mathbf{in} \ c, q) \Leftrightarrow (\mathbf{wp}(c, q) / v \rightarrow e)}$$

when v is not free in q .

All of the command-specific rules for **wp** also hold for **wlp**.

There are no similar rules for **while** commands. This is why it is necessary to provide the invariants of **while** commands (and the variants, in the case of total correctness) in order to mechanically generate verification conditions.

Weakest Preconditions versus Annotated Specifications

Sometimes weakest precondition reasoning requires fewer intermediate assertions than annotated specifications. For example, to prove

$$\begin{array}{l} \{x = 3 \vee y = 3\} \\ z := y ; \\ \mathbf{if\ } z = 3 \mathbf{\ then\ } x := 3 \mathbf{\ else\ } y := 3 \\ \{x = 3 \wedge y = 3\} \end{array}$$

using annotated specifications requires an intermediate assertion:

$$\begin{array}{l} \{x = 3 \vee y = 3\} \\ z := y ; \\ \{(x = 3 \vee y = 3) \wedge z = y\} \\ \mathbf{if\ } z = 3 \mathbf{\ then\ } x := 3 \mathbf{\ else\ } y := 3 \\ \{x = 3 \wedge y = 3\}, \end{array}$$

which leads to three verification conditions

$$\begin{array}{l} (x = 3 \vee y = 3) \Rightarrow ((x = 3 \vee y = 3) \wedge y = y) \\ ((x = 3 \vee y = 3) \wedge z = y \wedge z = 3) \Rightarrow (3 = 3 \wedge y = 3) \\ ((x = 3 \vee y = 3) \wedge z = y \wedge \neg z = 3) \Rightarrow (x = 3 \wedge 3 = 3). \end{array}$$

Weakest Preconditions versus Annotated Specifications (continued)

On the other hand, using weakest preconditions,

$$\mathbf{wp}(z := y ; \mathbf{if } z = 3 \mathbf{ then } x := 3 \mathbf{ else } y := 3, x = 3 \wedge y = 3)$$

$$\Leftrightarrow \mathbf{wp}(z := y, \mathbf{wp}(\mathbf{if } z = 3 \mathbf{ then } x := 3 \mathbf{ else } y := 3, x = 3 \wedge y = 3))$$

$$\Leftrightarrow \mathbf{wp}(z := y, (z = 3 \Rightarrow \mathbf{wp}(x := 3, x = 3 \wedge y = 3)) \wedge (\neg z = 3 \Rightarrow \mathbf{wp}(y := 3, x = 3 \wedge y = 3)))$$

$$\Leftrightarrow \mathbf{wp}(z := y, (z = 3 \Rightarrow 3 = 3 \wedge y = 3) \wedge (\neg z = 3 \Rightarrow x = 3 \wedge 3 = 3))$$

$$\Leftrightarrow (y = 3 \Rightarrow 3 = 3 \wedge y = 3) \wedge (\neg y = 3 \Rightarrow x = 3 \wedge 3 = 3),$$

which leads to the single verification condition:

$$(x = 3 \vee y = 3) \Rightarrow (y = 3 \Rightarrow 3 = 3 \wedge y = 3) \wedge (\neg y = 3 \Rightarrow x = 3 \wedge 3 = 3).$$

The Rule of Continuity

- Continuity

$$\frac{\forall v. v \geq 0 \Rightarrow (q \Rightarrow (q/v \rightarrow v + 1))}{\mathbf{wp}(c, \exists v. v \geq 0 \wedge q) \Leftrightarrow \exists v. v \geq 0 \wedge \mathbf{wp}(c, q)}$$

when v does not occur free in c .

This rule holds if we exclude the uninitialized variable declaration. However, if we include this or any other language construct that permits unbounded nondeterminism, the rule is no longer sound.

For example, take v to be v , c to be the command

newvar y **in** $x := |y|$

(which sets x to an arbitrary nonnegative integer), and q to be $v \geq x$. Then the premise of the continuity rule is valid, and

$$\mathbf{wp}(\mathbf{newvar} \ y \ \mathbf{in} \ x := |y|, \exists v. v \geq 0 \wedge v \geq x)$$

is true, but

$$\exists v. v \geq 0 \wedge \mathbf{wp}(\mathbf{newvar} \ y \ \mathbf{in} \ x := |y|, v \geq x)$$

is false.