Class Notes for CS 819B - Spring 2004

1. Expressions, Commands, Assertions, and Specifications

John C. Reynolds
Department of Computer Science
Carnegie Mellon University

Revised January 23, 2004

Expressions

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

$$
\begin{aligned}
\langle \text{boolexp} \rangle &::= \mathbf{true} \mid \mathbf{false} \\
&\mid \langle \text{exp} \rangle = \langle \text{exp} \rangle \mid \langle \text{exp} \rangle \neq \langle \text{exp} \rangle \mid \cdots \\
&\mid \neg \langle \text{boolexp} \rangle \mid \langle \text{boolexp} \rangle \wedge \langle \text{boolexp} \rangle \mid \langle \text{boolexp} \rangle \vee \langle \text{boolexp} \rangle \\
&\mid \langle \text{boolexp} \rangle \Rightarrow \langle \text{boolexp} \rangle \mid \langle \text{boolexp} \rangle \Leftrightarrow \langle \text{boolexp} \rangle
\end{aligned}
$$

Semantics of Expressions

$$
\begin{aligned}
\text{States} &= \langle \text{var} \rangle \to \mathbf{Z} \\
[\![-]\!]_{\text{exp}} &\in \langle \text{exp} \rangle \to \text{States} \to \mathbf{Z} \\
[\![-]\!]_{\text{boolexp}} &\in \langle \text{boolexp} \rangle \to \text{States} \to \mathbf{B}.
\end{aligned}
$$

For example:

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

Commands

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

Structural Operational Semantics of Commands

An execution (of a command) is a sequence

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

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

A configuration is either:

- *nonterminal*: A command-state pair c, s,
- *terminal*: A state s.

For example:

$$\begin{gathered}\mathsf{x} := \mathsf{x} + 1 \,;\, \mathsf{y} := \mathsf{y} + \mathsf{x} \,;\, \mathsf{x} := \mathsf{x} + 1, [\,\mathsf{x}\colon 3 \mid \mathsf{y}\colon 7\,] \rightsquigarrow\\ \mathsf{y} := \mathsf{y} + \mathsf{x} \,;\, \mathsf{x} := \mathsf{x} + 1, [\,\mathsf{x}\colon 4 \mid \mathsf{y}\colon 7\,] \rightsquigarrow\\ \mathsf{x} := \mathsf{x} + 1, [\,\mathsf{x}\colon 4 \mid \mathsf{y}\colon 11\,] \rightsquigarrow\\ [\,\mathsf{x}\colon 5 \mid \mathsf{y}\colon 11\,].\end{gathered}$$

Structural Operational Semantics (continued)

We write:

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

Then our semantics is *total*:

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

Except for indeterminate initialization of local variables, our present semantics is *determinate*:

- For every nonterminal configuration γ, there is exactly one configuration γ' such that $\gamma \leadsto \gamma'$.
- Every execution sequence can be extended in exactly one way to a sequence that is either infinite or ends with a terminal configuration.

Assertions

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

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. Note that assertions include boolean expressions ($\langle\text{boolexp}\rangle$).

Semantics of Assertions

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

In place of $[\![p]\!]_{\text{assert}}s$, we write $s \vDash p$.

For example:

$$\begin{aligned}s \vDash \mathsf{x} + \mathsf{y} = \mathsf{x} \ &\text{iff}\ \ [\![\mathsf{x} + \mathsf{y}]\!]_{\text{exp}}s = [\![\mathsf{x}]\!]_{\text{exp}}s\\ &\text{iff}\ \ s\,\mathsf{y} = 0.\end{aligned}$$

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

where $[\, s \mid x\colon n \,]\, y = \textbf{if}\ x = y\ \textbf{then}\ n\ \textbf{else}\ s(y)$.

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 $\mathcal{M}_{\mathrm{int}}$ of the integers. In place of $\mathcal{M}_{\mathrm{int}} \vDash_s p$ we write $s \vDash p$, and we call s a *state* or *store*, rather than an *assignment* or *environment*.

- Expressions in the programming language are the same as terms in the predicate calculus. They always terminate, never give error stops, and never have side effects.

  Thus, for example, there will be problems with division by zero if we introduce division as an arithmetic operator. More seriously, there will be problems with array designators that cause subscript errors, which we will address in the next chapter.

Some Terminology

- When $s \models p$ for some $s \in$ States, we say
  - p is *true* in s,
  - p *holds* for s,
  - p *describes* s,
  - s *satisfies* p.
- When $s \models p$ for all states $s \in$ States, we say
  - p is *valid*.
- When $s \models p$ for none of the states $s \in$ States, we say
  - p is *unsatisfiable*.
- When $s \models q$ holds for every state $s \in$ States 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.)
- When $s \models q$ holds for exactly those states $s \in$ States such that $s \models p$, we say
  - p is *equivalent* to q.

Binding and Free Variables

There are four binding constructions in our languages. For each of them, we define a variable occurrence called a *binder* (or *binding occurrence*), and a subphase called its *scope*:

1. **newvar** v **in** c, where v is a binder whose scope is c,

2. **newvar** $v := e$ **in** c, where v is a binder whose scope is c,

3. $\forall v.\ p$, where v is a binder whose scope is p,

4. $\exists v.\ p$, where v is a binder whose scope is p.

- An occurrence of a variable v in a phrase p is said to be *free* iff p does not contain a binder of v whose scope contains the occurrence.

- Otherwise, the occurrence is said to be bound by the binder of v whose scope includes the occurrence.

- If several binders of v have a scope including the occurrence, than the occurrence is bound by the one of these binders that has the smallest scope.

We write $\mathrm{FV}(p)$ for the set of variables with free occurrences in the phrase p.

Renaming

One can *rename* a bound variable v by replacing a binder of v, along with all occurrences of v bound by that binder, by some other variable that does not occur in the scope of the binder. Renaming does not change the meaning of any command or assertion.

Substitution

For any phrase p, we write

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

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 if necessary to avoid capture, i.e., to make each e_i free for v_i in p.

Modified (Free Assigned) Variables of a Command

$$\mathrm{MD}(c) = \text{“The free variables assigned by } c\text{”}$$

$$\begin{aligned}
\mathrm{MD}(v := e) &= \{v\} \\
\mathrm{MD}(\mathbf{skip}) &= \{\} \\
\mathrm{MD}(c_0\, ;\, c_1) &= \mathrm{MD}(c_0) \cup \mathrm{MD}(c_1) \\
\mathrm{MD}(\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1) &= \mathrm{MD}(c_0) \cup \mathrm{MD}(c_1) \\
\mathrm{MD}(\mathbf{while}\ b\ \mathbf{do}\ c) &= \mathrm{MD}(c) \\
\mathrm{MD}(\mathbf{newvar}\ v\ \mathbf{in}\ c) &= \mathrm{MD}(c) - \{v\} \\
\mathrm{MD}(\mathbf{newvar}\ v := e\ \mathbf{in}\ c) &= \mathrm{MD}(c) - \{v\}
\end{aligned}$$

Specifications (Hoare Triples)

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

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

When $[\![sp]\!]_{\text{spec}}$ is true we say that sp is *valid*.

Partial correctness:

$$
\begin{array}{l}
\{p\}\ c\ \{q\} \text{ holds iff } \forall s \in \text{States. } s \vDash p \text{ implies} \\
\quad\quad \forall s' \in \text{States. } c, s \rightsquigarrow^* s' \text{ implies } s' \vDash q
\end{array}
$$

Total correctness:

$$
\begin{array}{l}
[\ p\ ]\ c\ [\ q\ ] \text{ holds iff } \forall s \in \text{States. } s \vDash p \text{ implies} \\
\quad\quad \neg\,(c, s \uparrow) \\
\quad\quad \text{and } \forall s' \in \text{States. } c, s \rightsquigarrow^* s' \text{ implies } s' \vDash q
\end{array}
$$

Examples of Valid Partial Correctness Specifications

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

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

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

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

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

Examples of Valid Total Correctness Specifications

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

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

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

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:

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

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

Notice that the requirement that premisses that are assertions must be valid (not merely true in states satisfying certain assumptions) is different from natural deduction.

Inference Rules for Specifications:

Assignment (AS)

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

Instances

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

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

Inference Rules for Specifications:

Sequential Composition (SQ)

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

An Instance

$$\frac{\begin{array}{r}\{(2\times \mathsf{y})-\mathsf{y}>3\}\ \mathsf{x}:=2\times\mathsf{y}\ \{\mathsf{x}-\mathsf{y}>3\}\\ \{\mathsf{x}-\mathsf{y}>3\}\ \mathsf{x}:=\mathsf{x}-\mathsf{y}\ \{\mathsf{x}>3\}\end{array}}{\{(2\times\mathsf{y})-\mathsf{y}>3\}\ \mathsf{x}:=2\times\mathsf{y}\,;\mathsf{x}:=\mathsf{x}-\mathsf{y}\ \{\mathsf{x}>3\}}$$

Note

Except for **while** commands and recursive procedures, reasoning for partial and total correctness 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)

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

An Instance

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

Weakening Consequent (WC)

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

An Instance

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

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

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

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

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

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

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

A More Realistic Proof (continued)

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

*since

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

Inference Rules for Specifications:

An Alternative "Forward" Rule for Assignment (ASALT)

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

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

$$\{\mathsf{f} = \mathrm{fib}(\mathsf{k}) \land \mathsf{g} = \mathrm{fib}(\mathsf{k} - 1) \land \mathsf{k} \le \mathsf{n} \land \mathsf{k} \ne \mathsf{n}\}$$
$$\mathsf{t} := \mathsf{g}\ ;\ \mathsf{g} := \mathsf{f}\ ;\ \mathsf{f} := \mathsf{f} + \mathsf{t}\ ;\ \mathsf{k} := \mathsf{k} + 1$$
$$\{\mathsf{f} = \mathrm{fib}(\mathsf{k}) \land \mathsf{g} = \mathrm{fib}(\mathsf{k} - 1) \land \mathsf{k} \le \mathsf{n}\},$$

we get the verification condition

$$\begin{aligned}&(\exists \mathsf{k}'.\ \exists \mathsf{f}'.\ \exists \mathsf{g}'.\ \mathsf{k} = \mathsf{k}' + 1 \land \mathsf{f} = \mathsf{f}' + \mathsf{t} \land \mathsf{g} = \mathsf{f}' \land \mathsf{t} = \mathsf{g}'\\ &\qquad\qquad \land \mathsf{f}' = \mathrm{fib}(\mathsf{k}') \land \mathsf{g}' = \mathrm{fib}(\mathsf{k}' - 1) \land \mathsf{k}' \le \mathsf{n} \land \mathsf{k}' \ne \mathsf{n}) \Rightarrow\\ &\qquad \mathsf{f} = \mathrm{fib}(\mathsf{k}) \land \mathsf{g} = \mathrm{fib}(\mathsf{k} - 1) \land \mathsf{k} \le \mathsf{n}.\end{aligned}$$

Inference Rules for Specifications:

Partial Correctness of **while** (WHP)

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

Here i is the *invariant*.

An Instance

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

Inference Rules for Specifications:

Total Correctness of **while** (WHT)

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

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

An Instance

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

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

$$
\begin{array}{l}
[\mathbf{true}] \\
\mathbf{while}\ \mathsf{x} \neq 0\ \mathbf{do} \\
\quad \mathbf{if}\ \mathsf{x} < 0\ \mathbf{then}\ (\mathbf{newvar}\ \mathsf{y}\ \mathbf{in}\ \mathsf{x} := |\mathsf{y}|)\ \mathbf{else}\ \mathsf{x} := \mathsf{x} - 1 \\
[\mathsf{x} = 0]
\end{array}
$$

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

This program contains the command

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

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

Inference Rules for Specifications:

skip (SK)

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

Conditional (CD)

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

An Alternative Rule for Conditionals (CDALT)

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

Inference Rules for Specifications:

Variable Declaration (DC)

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

when v does not occur free in p or q.

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

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

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

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

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

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

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. To determine this reconstruction, we must use rules in which the premises are completely determined by the conclusion.

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

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

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

Annotated Specifications (continued)

But if we change the rule to

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

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

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

Then, once q is determined, the premisses must be

$$\begin{array}{l} \{\mathsf{n} \geq 0\} \\ (\mathsf{k} := 0\ ;\mathsf{s} := 1); \\ \{\mathsf{s} = 2^{\mathsf{k}} \wedge \mathsf{k} \leq \mathsf{n}\} \end{array}$$

and

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

Rules for Annotated Specifications

To obtain annotated specifications, we must change the following rules:

Sequential Composition (SQAN)

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

Strengthening Precedent (SPAN)

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

Weakening Consequent (WCAN)

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

Total Correctness of **while**

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

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

More Rules for Annotated Specifications

To reduce the number of unnecessary annotations, it is useful to change further rules. (Each of the following rules can be derived from earlier rules.)

Assignment (ASAN)

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

skip (SKAN)

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

Partial Correctness of **while** (WHPAN)

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

Total Correctness of **while** (WHTAN)

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

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

An Annotated Specification for Fibonacci Numbers

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

Verification Conditions

1. $\mathsf{n} \geq 0 \wedge \mathsf{n} = 0 \Rightarrow 0 = \mathrm{fib}(\mathsf{n})$
2. $\mathsf{n} \geq 0 \wedge \neg\, \mathsf{n} = 0 \Rightarrow 1 = \mathrm{fib}(1) \wedge 0 = \mathrm{fib}(1-1) \wedge 1 \leq \mathsf{n}$
3. $\mathsf{f} = \mathrm{fib}(\mathsf{k}) \wedge \mathsf{g} = \mathrm{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n} \wedge \mathsf{k} \neq \mathsf{n} \wedge \mathsf{n} - \mathsf{k} = \mathsf{v}_0 \Rightarrow$
   $\mathsf{f} + \mathsf{g} = \mathrm{fib}(\mathsf{k}+1) \wedge \mathsf{f} = \mathrm{fib}((\mathsf{k}+1)-1) \wedge \mathsf{k}+1 \leq \mathsf{n}$
   $\wedge\, \mathsf{n} - (\mathsf{k}+1) < \mathsf{v}_0$
4. $\mathsf{f} = \mathrm{fib}(\mathsf{k}) \wedge \mathsf{g} = \mathrm{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n} \wedge \mathsf{k} \neq \mathsf{n} \Rightarrow \mathsf{n} - \mathsf{k} \geq 0$
5. $\mathsf{f} = \mathrm{fib}(\mathsf{k}) \wedge \mathsf{g} = \mathrm{fib}(\mathsf{k}-1) \wedge \mathsf{k} \leq \mathsf{n} \wedge \neg\, \mathsf{k} \neq \mathsf{n} \Rightarrow \mathsf{f} = \mathrm{fib}(\mathsf{n})$

An Annotated Specification for Fast Exponentiation

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

Verification Conditions

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

2. $\mathsf{y} \times \mathsf{z}^{\mathsf{k}} = \mathsf{x}^{\mathsf{n}} \wedge \mathsf{k} \geq 0 \wedge \mathsf{k} \neq 0 \wedge \mathsf{k} = \mathsf{k}_0 \wedge \textbf{odd}(\mathsf{k}) \Rightarrow$
$$(\mathsf{y} \times \mathsf{z}) \times (\mathsf{z} \times \mathsf{z})^{(\mathsf{k}-1)\div 2} = \mathsf{x}^{\mathsf{n}}$$
$$\wedge\ (\mathsf{k} - 1) \div 2 \geq 0 \wedge (\mathsf{k} - 1) \div 2 < \mathsf{k}_0$$

3. $\mathsf{y} \times \mathsf{z}^{\mathsf{k}} = \mathsf{x}^{\mathsf{n}} \wedge \mathsf{k} \geq 0 \wedge \mathsf{k} \neq 0 \wedge \mathsf{k} = \mathsf{k}_0 \wedge \neg\, \textbf{odd}(\mathsf{k}) \Rightarrow$
$$\mathsf{y} \times (\mathsf{z} \times \mathsf{z})^{\mathsf{k}\div 2} = \mathsf{x}^{\mathsf{n}} \wedge \mathsf{k} \div 2 \geq 0 \wedge \mathsf{k} \div 2 < \mathsf{k}_0$$

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

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

Interderivability of the Assignment Rules

$$\frac{}{\{p/v \to e\}\ v := e\ \{p\}}$$

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

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

To show the second rule, assuming the first:

$\{p\}$
$\{e = e \wedge p\}$
$\{(e = e' \wedge p')/v' \to v\}$
$\{\exists v'.\ e = e' \wedge p'\}$
$v := e$
$\{\exists v'.\ v = e' \wedge p'\}$

To show the first rule, assuming the second:

$\{p/v \to e\}$
$v := e$
$\{\exists v'.\ v = e' \wedge ((p/v \to e)/v \to v')\}$
$\{\exists v'.\ v = e' \wedge (p/v \to e')\}$
$\{\exists v'.\ v = e' \wedge p\}$
$\{\exists v'.\ p\}$
$\{p\}$

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

Interderivability of the Conditional Rules

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

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

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:

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

Interderivability of the Conditional Rules (continued)

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

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

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

$\{p\}$
$\{(b \Rightarrow p \wedge b) \wedge (\neg\ b \Rightarrow p \wedge \neg\ b)\}$
$\mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2$
$\{q\}$,

which is the conclusion of the second rule.

Structural Inference Rules

(In addition to Strengthening Precedent and Weakening Consequent)

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.

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

Substitution (SUB)

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

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

For example, in

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

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

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

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

Structural Inference Rules:

Disjunction (DISJ)

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

Conjunction (CONJ)

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

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.

Structural Inference Rules:

Constancy (CONST)

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

when no variable occurs both free in r and assigned 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 occurs both free in r and assigned 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 states 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:

$$
\begin{array}{l}
[\mathsf{k} \geq 0 \land \mathsf{k} = \mathsf{k}_0] \\
\mathbf{newvar}\ \mathsf{z}\ \mathbf{in} \\
\quad \big(\mathsf{z} := \mathsf{x}\,;\ \mathsf{y} := 1\,; \\
\quad [\mathsf{y} \times \mathsf{z}^{\mathsf{k}} = \mathsf{x}^{\mathsf{k}_0} \land \mathsf{k} \geq 0] \\
\quad \mathbf{while}\ [\mathbf{vrnt}\colon \mathsf{k}]\ \mathsf{k} \neq 0\ \mathbf{do} \\
\qquad (\mathbf{if}\ \mathbf{odd}(\mathsf{k})\ \mathbf{then}\ (\mathsf{k} := \mathsf{k} - 1\,;\ \mathsf{y} := \mathsf{y} \times \mathsf{z})\ \mathbf{else}\ \mathbf{skip}\,; \\
\qquad \mathsf{k} := \mathsf{k} \div 2\,;\ \mathsf{z} := \mathsf{z} \times \mathsf{z})\big) \\
[\mathsf{y} = \mathsf{x}^{\mathsf{k}_0}]
\end{array}
$$

Totality is built into 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:

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

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

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

Even in the absence of any axionms 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.

We will see how to get around this problem in the next chapter.

Varied Arithmetic

In general, integer division satisfies

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

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

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

a few machines provide a "number-theoretic" division satisfying

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

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:

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