Appendix C

Summary of laws

The laws, definitions, and abbreviations appear below in alphabetical order.

Law 1.8 <u>*absorb assumption*</u> *p.12*

An assumption before a specification can be absorbed directly into its precondition.

$$\{pre'\}\ w{:}\,[pre\ ,\ post] \quad = \quad w{:}\,[pre' \wedge pre\ ,\ post]\ .$$

□

Law 17.2 <u>*absorb coercion*</u> *p.165*

A coercion following a specification can be absorbed into its postcondition.

$$w{:}\,[pre\ ,\ post]\,;\ [post'] \quad = \quad w{:}\,[pre\ ,\ post \wedge post']\ .$$

□

Law 22.2 <u>*advance assumption*</u> *p.222*

$$w := E\ \{pre\} \quad = \quad \{pre[w\backslash E]\}\ w := E\ .$$

□

Law 22.1 <u>*advance coercion*</u> *p.222*

$$w := E\ \ [post] \quad = \quad [post[w\backslash E]]\ \ w := E\ .$$

□

Law 4.1 <u>*alternation*</u> *p.37*

If $pre \Rrightarrow GG$, then

$$\begin{array}{ll} & w{:}\,[pre\ ,\ post] \\ \sqsubseteq & \mathbf{if}\ ([\!]\ i \cdot G_i \rightarrow w{:}\,[G_i \wedge pre\ ,\ post])\ \mathbf{fi}\ . \end{array}$$

□

Law 4.2 *alternation* *p.39*

$$\begin{array}{ll} & \{(\bigvee i \cdot G_i)\}\ prog \\ = & \textbf{if}\ ([\!]\ i \cdot G_i \rightarrow \{G_i\}\ prog)\ \textbf{fi}\ . \end{array}$$

□

Law 4.3 *alternation guards* *p.40*

Let GG mean $G_0 \vee \cdots \vee G_n$, and HH similarly. Then provided

1. $GG \Rrightarrow HH$, and
2. $GG \Rrightarrow (H_i \Rightarrow G_i)$ for each i separately,

this refinement is valid:

$$\textbf{if}\ ([\!]\ i \cdot G_i \rightarrow prog_i)\ \textbf{fi} \quad \sqsubseteq \quad \textbf{if}\ ([\!]\ i \cdot H_i \rightarrow prog_i)\ \textbf{fi}\ .$$

□

Law 5.2 *assignment* *p.44*

If $(w = w_0) \wedge (x = x_0) \wedge pre \Rrightarrow post[w \backslash E]$, then

$$w, x\colon [pre\ ,\ post] \quad \sqsubseteq \quad w := E\ .$$

□

Definition 23.1 *assignment* *p.250*

For any postcondition $\mathcal{A}$,

$$wp(w := E, \mathcal{A})\ \widehat{=}\ \mathcal{A}[w \backslash E]\ .$$

□

Law 1.3 *assignment* *p.8*

If $pre \Rrightarrow post[w \backslash E]$, then

$$w, x\colon [pre\ ,\ post] \quad \sqsubseteq \quad w := E\ .$$

□

Abbreviation 1.6 *assumption* *p.11*

$$\{pre\}\ \widehat{=}\colon [pre\ ,\ \mathsf{true}]\ .$$

□

Definition 23.7 *assumption* *p.254*

$$wp(\{pre\}, \mathcal{A})\ \widehat{=}\ pre \wedge \mathcal{A}\ .$$

□

Law 17.7 *augment assignment* *p.167*
The assignment $w := E$ can be replaced by the fragment

$$\{CI\}\ w, c := E, ?\ [CI]\ .$$

□

Law 17.8 *augment assignment* *p.167*
The assignment $w := E$ can be replaced by the assignment $w, c := E, F$ provided that

$$CI \quad \Rrightarrow \quad CI[w, c \backslash E, F]\ .$$

□

Law B.3 *augment assumption* *p.289*
The assumption $\{pre\}$ *becomes* $\{pre \wedge CI\}$. □

Law B.5 *augment coercion* *p.289*
The coercion $[post]$ *becomes* $[CI \Rightarrow post]$.
□

Law 17.9 *augment guard* *p.168*
The guard G may be replaced by G' provided that

$$CI \quad \Rrightarrow \quad (G \Leftrightarrow G')\ .$$

□

Law 17.5 *augment initialisation* *p.166*
The initialisation I *becomes* $I \wedge CI$.
□

Law 17.6 *augment specification* *p.167*
The specification $w\colon [pre\ ,\ post]$ *becomes*

$$w, c\colon [pre\ ,\ CI\ ,\ post]\ .$$

□

Abbreviation 17.1 *coercion* *p.165*
no initial variables,

$$[post]\ \widehat{=}\ \colon [\mathsf{true}\ ,\ post]\ .$$

□

Definition 23.8 *coercion* *p.254*
no initial variables,

$$wp([post], \mathcal{A})\ \widehat{=}\ post \Rightarrow \mathcal{A}\ .$$

□

Law 18.7 *collapse identical branches* *p.194*

$$\begin{array}{ll} & \textbf{if } ([\!]\ i \cdot G_i \rightarrow branch_i) \\ & [\!]\ \ G \rightarrow branch \\ & [\!]\ \ G' \rightarrow branch \\ & \textbf{fi} \\ = & \textbf{if } ([\!]\ i \cdot G_i \rightarrow branch_i) \\ & [\!]\ \ G \vee G' \rightarrow branch \\ & \textbf{fi}\ . \end{array}$$

□

Law 5.4 *contract frame* *p.45*

$$w, x\colon [pre\ ,\ post] \quad \sqsubseteq \quad w\colon [pre\ ,\ post[x_0 \backslash x]]\ .$$

□

Definition 23.13 *data refinement* *p.256*

Let a be a list of variables called *abstract*, let c be a list of variables called *concrete*, and let I be a formula called the *coupling invariant*. Then command *progA* is data-refined to *progC* by a, c, I exactly when for all postconditions $\mathcal{A}$ not containing c we have

$$(\exists\, a \cdot I \wedge wp(progA, \mathcal{A})) \quad \Rrightarrow \quad wp(progC, (\exists\, a \cdot I \wedge \mathcal{A}))\ .$$

□

Law 17.16 *data-refine assignment* *p.177*

Under abstraction function af and data-type invariant dti, the assignment $w, a := E, F$ can be replaced by the assignment $w, c := E[a \backslash\ \mathsf{af}\ c], G$ provided that G contains no a, and that

$$\begin{array}{rl} & \mathsf{dti}\ c \Rrightarrow F[a \backslash\ \mathsf{af}\ c] = \mathsf{af}\ G \\ \text{and} & \mathsf{dti}\ c \Rrightarrow \mathsf{dti}\ G\ . \end{array}$$

□

Law 17.17 *data-refine guard* *p.178*

Under abstraction function af and data-type invariant dti, the guard G may be replaced by $G[a \backslash\ \mathsf{af}\ c] \wedge \mathsf{dti}\ c$, or if desired simply by $G[a \backslash\ \mathsf{af}\ c]$ on its own.

□

Law 17.14 *data-refine initialisation* *p.175*

Under abstraction function af and data-type invariant dti, the initialisation I *becomes*

$$I[a \backslash\ \mathsf{af}\ c] \wedge \mathsf{dti}\ c\ .$$

□

Law 17.15 <u>*data-refine specification*</u> *p.176*
Under abstraction function af and data-type invariant dti, the specification $w, a\colon [pre\ ,\ post]$ *becomes*

$$w, c\colon [pre[a \backslash \mathsf{af}\ c]\ ,\ \mathsf{dti}\ c\ ,\ post[a_0, a \backslash \mathsf{af}\ c_0, \mathsf{af}\ c]]\ .$$

□

Abbreviation 1.5 <u>*default precondition*</u> *p.11*

$$w\colon [post]\ \widehat{=}\ w\colon [\mathsf{true}\ ,\ post]\ .$$

□

Law 17.13 <u>*diminish assignment*</u> *p.170*
If E contains no variables a, then the assignment $w, a := E, F$ can be replaced by the assignment $w := E$.
□

Law B.4 <u>*diminish assumption*</u> *p.289*
The assumption $\{pre\}$ is diminished to $\{(\exists\, a : A \bullet pre)\}$.
□

Law B.6 <u>*diminish coercion*</u> *p.290*
The coercion $[post]$ *becomes* $[(\forall\, a : A \bullet post)]$.
□

Law 17.10 <u>*diminish initialisation*</u> *p.168*
The initialisation I *becomes*

$$(\exists\, a : A \bullet I)\,.$$

□

Law 17.11 <u>*diminish specification*</u> *p.169*
The specification $w, a\colon [pre\ ,\ post]$ *becomes*

$$w\colon [(\exists\, a : A \bullet pre)\ ,\ (\forall\, a_0 : A \bullet pre_0 \Rightarrow (\exists\, a : A \bullet post))]\ ,$$

where pre_0 is $pre[w, a \backslash w_0, a_0]$. The frame beforehand *must* include a.
□

Law 17.12 <u>*diminish specification*</u> *p.169*
The specification $w\colon [pre\ ,\ post]$ *becomes*

$$w\colon [(\exists\, a : A \bullet pre)\ ,\ (\forall\, a : A \bullet pre_0 \Rightarrow post)]\ ,$$

where pre_0 is $pre[w \backslash w_0]$. The frame beforehand need not include a, and *post* must not contain a_0.
□

Law 17.4 *establish assumption* *p.166*
An assumption after a specification can be removed after suitable strengthening of the precondition.

$$\begin{aligned} & w\colon [pre\ ,\ post]\,;\ \{pre'\} \\ = \ & w\colon [pre \wedge (\forall\, w \cdot post \Rightarrow pre')\,[w_0 \backslash w]\ ,\ post]\ . \end{aligned}$$

□

Law 8.3 *expand frame* *p.69*

$$w\colon [pre\ ,\ post] \quad = \quad w, x\colon [pre\ ,\ post \wedge x = x_0]\ .$$

□

Law 8.6 *expand frame* *p.72*
For fresh constant X,

$$\begin{aligned} & w\colon [pre\ ,\ post] \\ \sqsubseteq\ & \mathbf{con}\ X \cdot \\ & w, x\colon [pre \wedge x = X\ ,\ post \wedge x = X]\,. \end{aligned}$$

□

Definition 23.6 *feasibility* *p.254*
Command *prog* is *feasible* exactly when

$$wp(prog, \mathsf{false}) \quad \equiv \quad \mathsf{false}\ .$$

Otherwise it is *infeasible.*
□

Definition 1.4 *feasibility* *p.10*
The specification $w\colon [pre\ ,\ post]$ is *feasible* iff

$$pre \quad \Rrightarrow \quad (\exists\, w : T \cdot post)\ ,$$

where T is the type[a] of the variables w.
□

[a]In Chapter 6 the notion of type will be generalised to include so-called 'local invariants', and then a more comprehensive definition (6.5) of feasibility will be appropriate. It does not concern us now, but must be borne in mind if ever referring to the definition above once local invariants have been introduced.

Definition 6.5 *feasibility* *p.58*
The specification $w\colon [pre\ ,\ post]$ is *feasible* in context *inv* iff

$$(w = w_0) \wedge pre \wedge inv \quad \Rrightarrow \quad (\exists\, w : T \cdot inv \wedge post)\,,$$

where T is the type of w.
□

Law 6.3 *fix initial value* *p.56*
For any term E such that $pre \Rrightarrow E \in T$, and fresh name c,

$$\begin{array}{ll} & w\colon [pre \;,\; post] \\ \sqsubseteq & \mathbf{con}\ c : T \bullet \\ & w\colon [pre \wedge c = E \;,\; post] \;. \end{array}$$

□

Law 18.6 *flatten nested alternations* *p.192*

$$\begin{array}{ll} & \mathbf{if}\ ([\!]\ i \cdot G_i \rightarrow \mathbf{if}\ ([\!]\ j \cdot H_j \rightarrow branch_{ij})\ \mathbf{fi})\ \mathbf{fi} \\ = & \mathbf{if}\ ([\!]\ i, j \cdot G_i \wedge H_j \rightarrow branch_{ij})\ \mathbf{fi} \;. \end{array}$$

□

Law 3.5 *following assignment* *p.32*
For any term E,

$$\begin{array}{ll} & w, x\colon [pre \;,\; post] \\ \sqsubseteq & w, x\colon [pre \;,\; post[x \backslash E]] \;; \\ & x := E \;. \end{array}$$

□

Abbreviation 8.2 *initial variable* *p.69*
Occurrences of 0-subscripted variables in the postcondition of a specification refer to values held by those variables in the *initial* state. Let x be any variable, probably occurring in the frame w. If X is a fresh name, and T is the type of x, then

$$\begin{array}{ll} & w\colon [pre \;,\; post] \\ \widehat{=} & |[\ \mathbf{con}\ X : T \bullet w\colon [pre \wedge x = X \;,\; post[x_0 \backslash X]]\ ]| \;. \end{array}$$

□

Law B.1 *initialised iteration* *p.272*
Provided *inv* contains no initial variables,

$$\begin{array}{ll} & w\colon [pre \;,\; inv \wedge \neg G] \\ \sqsubseteq & w\colon [pre \;,\; inv] \;; \\ & \mathbf{do}\ G \rightarrow w\colon [G \wedge inv \;,\; inv \wedge (0 \leq V < V_0)]\ \mathbf{od} \;. \end{array}$$

□

Law 17.19 *introduce assumption* *p.183*

$$[post] \quad \sqsubseteq \quad [post]\ \{post\}.$$

□

Law 17.3 <u>*introduce coercion*</u> *p.165*
skip is refined by any coercion.

$$\mathbf{skip} \quad \sqsubseteq \quad [post] \; .$$

□
Law 6.1 <u>*introduce local variable*</u> *p.55*
If x does not occur in w, *pre* or *post* then

$$w\colon [pre \;,\; post] \quad \sqsubseteq \quad |[\; \mathbf{var}\; x : T;\; \mathbf{and}\; inv \cdot w, x\colon [pre \;,\; post] \;]| \; .$$

□
Law 6.2 <u>*introduce logical constant*</u> *p.56*
If $pre \Rrightarrow (\exists\, c : T \cdot pre')$, and c does not occur in w, *pre* or *post*, then

$$\begin{array}{ll} & w\colon [pre \;,\; post] \\ \sqsubseteq & \mathbf{con}\; c : T \cdot \\ & w\colon [pre' \;,\; post] \; . \end{array}$$

If the optional type T is omitted, then the quantification in the proviso should be untyped.
□
Law 5.5 <u>*iteration*</u> *p.46*
Let *inv*, the *invariant*, be any formula; let V, the *variant*, be any integer-valued expression. Then if GG is the disjunction of the guards,

$$\begin{array}{ll} & w\colon [inv \;,\; inv \wedge \neg GG] \\ \sqsubseteq & \mathbf{do}\; ([\!]\, i \cdot G_i \rightarrow w\colon [inv \wedge G_i \;,\; inv \wedge (0 \leq V < V_0)])\; \mathbf{od} \; . \end{array}$$

Neither *inv* nor G_i may contain initial variables. The expression V_0 is $V[w \backslash w_0]$.
□
Law 8.5 <u>*leading assignment*</u> *p.71*
For any expression E,

$$\begin{array}{ll} & w, x\colon [pre[x \backslash E] \;,\; post[x_0 \backslash E_0]] \\ \sqsubseteq & x := E; \\ & w, x\colon [pre \;,\; post] \; . \end{array}$$

The expression E_0 abbreviates $E[w, x \backslash w_0, x_0]$.
□
Law 3.6 <u>*leading assignment*</u> *p.35*
For disjoint w and x,

$$w, x := E, F[w \backslash E] \quad = \quad w := E;\;\; x := F \; .$$

□

Law 18.2 *left-distribution of composition over alternation* *p.190*

$$\begin{array}{rl} & \textbf{if } ([\!]\ i \cdot G_i \rightarrow \mathit{branch}_i)\ \textbf{fi};\ \ \mathit{prog} \\ = & \textbf{if } ([\!]\ i \cdot G_i \rightarrow \mathit{branch}_i;\ \ \mathit{prog})\ \textbf{fi}\ . \end{array}$$

□

Abbreviation 18.1 *local block initialisation* *p.187*

$$\begin{array}{rl} & |[\ \textbf{var}\ l : T;\ \ \textbf{initially}\ \mathit{inv} \cdot \mathit{prog}\ ]| \\ \widehat{=} & |[\ \textbf{var}\ l : T \cdot l\text{: } [\mathit{inv}]\ ;\ \ \mathit{prog}\ ]|\ . \end{array}$$

□

Definition 23.4 *local variable* *p.253*

$$\mathit{wp}(|[\ \textbf{var}\ x \cdot \mathit{prog}\ ]|, \mathcal{A})\ \ \widehat{=}\ \ (\forall\, x \cdot \mathit{wp}(\mathit{prog}, \mathcal{A}))\,,$$

provided $\mathcal{A}$ contains no free x.
□

Definition 23.5 *logical constant* *p.253*

$$\mathit{wp}(|[\ \textbf{con}\ x \cdot \mathit{prog}\ ]|, \mathcal{A})\ \ \widehat{=}\ \ (\exists\, x \cdot \mathit{wp}(\mathit{prog}, \mathcal{A}))\,,$$

provided $\mathcal{A}$ contains no free x.
□

Law 1.9 *merge assumptions* *p.14*

$$\{\mathit{pre}'\}\ \{\mathit{pre}\} \quad = \quad \{\mathit{pre}' \wedge \mathit{pre}\}\ .$$

□

Law 17.18 *merge coercions* *p.183*

$$[\mathit{post}]\ [\mathit{post}'] \quad = \quad [\mathit{post} \wedge \mathit{post}']\ .$$

□

Law 3.1 *open assignment* *p.29*
For any expression F,

$$w, x := E, ? \quad \sqsubseteq \quad w, x := E, F\ .$$

□

Definition 23.12 *recursion* *p.255*
Let $\mathcal{C}(p)$ be a program fragment in which the name p appears. Then

$$\textbf{re}\ p \cdot \mathcal{C}(p)\ \textbf{er}$$

is the *least-refined* program *fix* such that $\mathcal{C}(\mathit{fix}) = \mathit{fix}$.
□

Law 16.1 <u>*refine initialisation*</u> *p.157*

If $init' \Rrightarrow init$, then

$$\textbf{initially } init \quad \sqsubseteq \quad \textbf{initially } init' \ .$$

□

Law 16.2 <u>*refine module*</u> *p.158*

Let E be the list of exported procedures from module M, I its imported procedures, and *init* its initialisation. A module M' refines M if the following three conditions are satisfied:

1. Its exported variables are unchanged.
2. Its exported procedures E' refine E.
3. Its initialisation $init'$ refines *init*.

In addition, the following changes may be made provided the three conditions above are not invalidated as a result:

1. Its imported variables' declarations are weakened.
2. Its imported procedures I' are refined by I.
3. An imported procedure I is replaced by a *local* (neither imported nor exported) procedure I' that refines I.

□

Definition 23.3 <u>*refinement*</u> *p.252*

For any commands *prog*1 and *prog*2, we say that *prog*1 is refined by *prog*2, writing $prog1 \sqsubseteq prog2$, exactly when for all postconditions $\mathcal{A}$ we have

$$wp(prog1, \mathcal{A}) \quad \Rrightarrow \quad wp(prog2, \mathcal{A}) \ .$$

□

Law 18.5 <u>*remove alternation*</u> *p.192*

$$\textbf{if } \mathsf{true} \rightarrow branch \textbf{ fi} \quad = \quad branch \ .$$

□

Law 1.10 <u>*remove assumption*</u> *p.15*

Any assumption is refined by **skip**.

$$\{pre\} \quad \sqsubseteq \quad \textbf{skip} \ .$$

□

Law 17.20 <u>*remove coercion*</u> *p.183*

$$\{pre\}\ [pre] \quad \sqsubseteq \quad \{pre\}.$$

□

Law 18.4 *remove false guard* *p.192*

$$
\begin{array}{ll}
 & \textbf{if } ([\!]\ i \cdot G_i \to branch_i) \\
 & [\!]\ \ \mathsf{false} \to branch \\
 & \textbf{fi} \\
= & \textbf{if } ([\!]\ i \cdot G_i \to branch_i)\ \textbf{fi}\ .
\end{array}
$$

□

Law 7.1 *remove invariant* *p.66*

Provided w does not occur in inv,

$$w\colon [pre\ ,\ inv\ ,\ post] \quad \sqsubseteq \quad w\colon [pre\ ,\ post]\ .$$

□

Law 6.4 *remove logical constant* *p.57*

If c occurs nowhere in program $prog$, then

$$|[\ \textbf{con}\ c : T \cdot prog\ ]| \quad \sqsubseteq \quad prog\ .$$

□

Law 11.3 *result assignment* *p.110*

Given a procedure declaration that refines

$$\textbf{procedure}\ Proc\ (\textbf{result}\ f : T)\ \ \widehat{=}\ \ w, f := E, F\ ,$$

with f not occurring in E or in F, we have the following refinement:

$$w, a := E, F \quad \sqsubseteq \quad Proc\ (a)\ .$$

Variables a and f need not be different from each other, but w must be disjoint from both.

□

Law 11.4 *result specification* *p.111*

Given a procedure declaration that refines

$$\textbf{procedure}\ Proc\ (\textbf{result}\ f : T)\ \ \widehat{=}\ \ w, f\colon [pre\ ,\ post[a \backslash f]]\ ,$$

with f not occurring in pre, and neither f nor f_0 occurring in $post$, we have the following refinement:

$$w, a\colon [pre\ ,\ post] \quad \sqsubseteq \quad Proc\ (a)\ .$$

Again, variables a and f need not be different from each other, but w must be disjoint from both.

□

Law 18.3 *right-distribution of assignment over alternation* *p.191*

$$\begin{array}{ll} & x := E;\ \textbf{if}\ ([]\ i \cdot G_i \to branch_i)\ \textbf{fi} \\ = & \textbf{if}\ ([]\ i \cdot G_i[x \backslash E] \to x := E;\ \ branch_i)\ \textbf{fi}\ . \end{array}$$

□

Law 18.8 *select true guard* *p.196*

$$\begin{array}{ll} & \textbf{if}\ ([]\ i \cdot G_i \to branch_i) \\ & []\ \ \mathsf{true} \to branch \\ & \textbf{fi} \\ \sqsubseteq & branch\ . \end{array}$$

□

Abbreviation 10.1 *sequence assignment* *p.99*

For any sequence as, if $0 \le i, j \le \#as$ then

$$as[i := E][j] \ \widehat{=}\ \begin{array}{rll} E & \text{when} & i = j \\ as[j] & \text{when} & i \neq j\ . \end{array}$$

□

Law 3.3 *sequential composition* *p.31*

For any[a] formula mid,

$$w\colon [pre\ ,\ post] \quad \sqsubseteq \quad w\colon [pre\ ,\ mid]\ ;\ \ w\colon [mid\ ,\ post]\ \ .$$

□

[a]Neither *mid* nor *post*, however, may contain the so-called 'initial variables' that are the subject of Chapter 8 to come. That does not at all concern us now, but must be remembered if ever referring to this law later, once they have been introduced. Law B.2 on page 275 is the most appropriate replacement for the more general case.

Law 8.4 *sequential composition* *p.70*

For fresh constants X,

$$\begin{array}{ll} & w, x\colon [pre\ ,\ post] \\ \sqsubseteq & \textbf{con}\ X \cdot \\ & x\colon [pre\ ,\ mid]\ ; \\ & w, x\colon [mid[x_0 \backslash X]\ ,\ post[x_0 \backslash X]]\ . \end{array}$$

The formula *mid* must not contain initial variables other than x_0.

□

Law B.2 *sequential composition* *p.275*

$$\begin{array}{ll} & w, x\colon [pre\ ,\ post] \\ \sqsubseteq & x\colon [pre\ ,\ mid]\,; \\ & w, x\colon [mid\ ,\ post]\,. \end{array}$$

The formula *mid* must not contain initial variables; and *post* must not contain x_0.
□

Abbreviation 8.1 *simple specification* *p.68*
For any relation $\odot$,

$$w\colon \odot E \quad = \quad w\colon [w \odot E_0]\,,$$

where E_0 is $E[w \backslash w_0]$.
□

Law 1.7 *simple specification* *p.11*
Provided E contains no w,

$$w := E \quad = \quad w\colon [w = E]\ .$$

If w and E are lists, then the formula $w = E$ means the equating of corresponding elements of the lists.
□

Law 3.2 *skip command* *p.30*
If $pre \Rrightarrow post$, then

$$w\colon [pre\ ,\ post] \quad \sqsubseteq \quad \mathbf{skip}\ .$$

□

Law 5.3 *skip command* *p.45*
If $(w = w_0) \wedge pre \Rrightarrow post$, then

$$w\colon [pre\ ,\ post] \quad \sqsubseteq \quad \mathbf{skip}\ .$$

□

Law 3.4 *skip composition* *p.31*
For any program *prog*,

$$prog;\ \mathbf{skip} \quad = \quad \mathbf{skip};\ prog \quad = \quad prog\ .$$

□

Definition 23.2 *specification* *p.251*

$$wp(w\colon [pre\ ,\ post]\,, \mathcal{A}) \ \widehat{=}\ pre \wedge (\forall\, w \bullet post \Rightarrow \mathcal{A})\,[v_0 \backslash v]\ ,$$

where the substitution $[v_0 \backslash v]$ replaces *all* initial variables by corresponding final variables.
□

Abbreviation 5.6 *specification invariant* *p.46*
Provided *inv* contains no initial variables,

$$w\colon [pre\ ,\ inv\ ,\ post] \;\widehat{=}\; w\colon [pre \wedge inv\ ,\ inv \wedge post]\ .$$

□
Law 1.1 *strengthen postcondition* *p.7*
If $post' \Rrightarrow post$, then

$$w\colon [pre\ ,\ post] \quad \sqsubseteq \quad w\colon [pre\ ,\ post'] .$$

□
Law 5.1 *strengthen postcondition* *p.44*
If $pre[w \backslash w_0] \wedge post' \Rrightarrow post$, then

$$w\colon [pre\ ,\ post] \quad \sqsubseteq \quad w\colon [pre\ ,\ post']\ .$$

□
Definition 23.10 *substitution by result* *p.255*

$$wp(prog[\mathbf{result}\ f \backslash a], \mathcal{A}) \;\widehat{=}\; (\forall f \cdot wp(prog, \mathcal{A}[a \backslash f]))\,,$$

provided f does not occur free in $\mathcal{A}$.
□
Definition 23.9 *substitution by value* *p.255*

$$wp(prog[\mathbf{value}\ f \backslash E], \mathcal{A}) \;\widehat{=}\; wp(prog, \mathcal{A})[f \backslash E]\ ,$$

provided f does not occur free in $\mathcal{A}$.
□
Definition 23.11 *substitution by value-result* *p.255*

$$wp(prog[\mathbf{value\ result}\ f \backslash a], \mathcal{A}) \;\widehat{=}\; wp(prog, \mathcal{A}[a \backslash f])[f \backslash a],$$

provided f does not occur free in $\mathcal{A}$.
□

Law 15.1 *tagged alternation* *p.139*
Let first, middle and last be tags from a typical type declaration

$$\mathsf{first}\ A \cdots H \mid \mathsf{middle}\ I \cdots P \mid \mathsf{last}\ Q \cdots Z\ .$$

Provided none of $a \cdots h, q \cdots z$ appear free in E or *prog*, this refinement is valid:

$$\begin{array}{ll} & \{E\ \mathsf{is\ first} \vee E\ \mathsf{is\ last}\}\ prog \\ \sqsubseteq & \mathbf{if}\ E\ \mathbf{is} \\ & \quad \mathsf{first}\ a \cdots h \rightarrow \{E = \mathsf{first}\ a \cdots h\}\ prog \\ & [\!]\ \ \mathsf{last}\ q \cdots z \ \rightarrow \{E = \mathsf{last}\ q \cdots z\}\ prog \\ & \mathbf{fi}\ . \end{array}$$

□

Law 15.2 *tagged iteration* *p.145*
Let first, middle and last be tags from a type declaration

$$Type \mathrel{\widehat{=}} \mathsf{first}\ A \cdots H \mid \mathsf{middle}\ I \cdots P \mid \mathsf{last}\ Q \cdots Y\ .$$

Provided none of $a \cdots h, q \cdots y$ appears free in z, inv, E, or V, this refinement is valid:

$$\begin{array}{ll} & z\colon [inv\ ,\ inv \wedge \neg(E\ \mathsf{is\ first} \vee E\ \mathsf{is\ last})] \\ \sqsubseteq & \mathbf{do}\ E\ \mathbf{is} \\ & \quad \mathsf{first}\ a \cdots h \rightarrow z\colon [E = \mathsf{first}\ a \cdots h\ ,\ inv\ ,\ V \otimes V_0] \\ & [\!]\ \ \mathsf{last}\ q \cdots y \rightarrow z\colon [E = \mathsf{last}\ q \cdots y\ ,\ inv\ ,\ V \otimes V_0] \\ & \mathbf{od}\ . \end{array}$$

The formula *inv* is the invariant, the expression V is the variant, and the relation $\otimes$ must be well-founded.

□

Law 11.1 *value assignment* *p.108*
Given a procedure declaration that refines

$$\mathbf{procedure}\ Proc\ (\mathbf{value}\ f : T) \mathrel{\widehat{=}} w, f := E, ?\ ,$$

we have the following refinement:

$$w := E[f \backslash A] \quad \sqsubseteq \quad Proc\ (A)\ .$$

The actual parameter A may be an expression, and it should have type T. (If it does not, the refinement remains valid but subsequent type checking will fail.) As usual, variables w and f must be disjoint.

□

Law 11.2 *value specification* *p.109*
Given a procedure declaration that refines

$$\textbf{procedure } Proc\ (\textbf{value } f : T)\ \widehat{=}\ w, f\colon [pre\ ,\ post]\ ,$$

with *post* containing no f (but possibly f_0), the following refinement is valid:

$$w\colon [pre[f \backslash A]\ ,\ post[f_0 \backslash A_0]] \quad \sqsubseteq \quad Proc\ (A)\ ,$$

where A_0 is $A[w \backslash w_0]$.
□

Law 11.5 *value-result assignment* *p.113*
Given a procedure declaration that refines

$$\textbf{procedure } Proc\ (\textbf{value result } f : T)\ \widehat{=}\ w, f := E, F\ ,$$

we have the following refinement:

$$w, a := E[f \backslash a], F[f \backslash a] \quad \sqsubseteq \quad Proc\ (a)\ .$$

□

Law 11.6 *value-result specification* *p.113*
Given a procedure declaration that refines

$$\textbf{procedure } Proc\ (\textbf{value result } f : T)\ \widehat{=}\ w, f\colon [pre\ ,\ post[a \backslash f]]\ ,$$

with *post* not containing f, we have the following refinement:

$$w, a\colon [pre[f \backslash a]\ ,\ post[f_0 \backslash a_0]] \quad \sqsubseteq \quad Proc\ (a)\ .$$

□

Law 1.2 *weaken precondition* *p.7*
If $pre \Rrightarrow pre'$, then

$$w\colon [pre\ ,\ post] \quad \sqsubseteq \quad w\colon [pre'\ ,\ post]\,.$$

□

References

[Abr87] J.-R. Abrial. Generalised substitutions. 26 Rue des Plantes, Paris 75014, France, 1987.

[Bac78] R.-J.R. Back. On the correctness of refinement steps in program development. Report A-1978-4, Department of Computer Science, University of Helsinki, 1978.

[Bac80] R.-J.R. Back. Correctness preserving program refinements: Proof theory and applications. Tract 131, Mathematisch Centrum, Amsterdam, 1980.

[Bac86] R. Backhouse. *Program Construction and Verification.* Prentice-Hall, 1986.

[Bac87] R.-J.R. Back. Procedural abstraction in the refinement calculus. Report Ser.A 55, Departments of Information Processing and Mathematics, Swedish University of Åbo, Åbo, Finland, 1987.

[Bac88] R.-J.R. Back. A calculus of refinements for program derivations. *Acta Informatica*, 25:593–624, 1988.

[Bir86] R.S. Bird. Transformational programming and the paragraph problem. *Science of Computer Programming*, 6:159–189, 1986.

[Boo82] H. Boom. A weaker precondition for loops. *ACM Transactions on Programming Languages and Systems*, 4:668–677, 1982.

[CU89] Wei Chen and J.T. Udding. Towards a calculus of data refinement. In J.L.A. van de Snepsheut, editor, *Lecture Notes in Computer Science 375: Mathematics of Program Construction.* Springer, June 1989.

[Der83] N. Dershowitz. *The Evolution of Programs.* Birkhäuser, 1983.

[DF88] E.W. Dijkstra and W.H.J. Feijen. *A Method of Programming.* Addison-Wesley, 1988.

[Dij76] E.W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, Englewood Cliffs, 1976.