Chapter 4

Linking Theories

In the preceding two chapters we have developed a theory of programming through a series of stages; each stage concentrates on a different and successively smaller class of predicate, defined by healthiness conditions of increasing strength. At each stage, Exercise 3.6.1 shows that the notations of the programming language conserve the healthiness properties of their operands. Thus each stage could be presented independently as a separate closed theory, with its own notations, definitions and theorems. Later, we could explore the mathematical relationships between theories and show that the set of predicates expressible in each theory is essentially just a subset of those of the previous theory. The subset relation is one of the simplest forms of linkage between theories, but it serves as a model for the more elaborate linkage methods introduced in this chapter.

The most general possible way of defining a link between theories is as a function L which maps all predicates from one theory into a subset of the predicates from the other. An example familiar in computing is a compiler, which translates a program in a high level language to one expressed in a lower level language, executed directly by the hardware of a machine. In addition to these practical purposes, the link can reveal a lot about the structure of the theories which it is used to compare.

We shall assume in this chapter that nearly all the theories are complete lattices, and we shall use the symbol $\sqsubseteq$ to stand for the ordering relation between their elements. The use of the form $P \sqsubseteq Q$ in place of $[Q \Rightarrow P]$ emphasises the purely algebraic nature of our definitions and theorems and their proofs; it will help to generalise the results to other kinds of lattice ordering.

In dealing with functions that link theories, we will use standard functional notations. Theories will be identified primarily by their sets of predicates, denoted by bold capitals, for example $\mathbf{S}$, $\mathbf{T}$, $\mathbf{U}$. A total function L which maps every element of $\mathbf{S}$ to some element of $\mathbf{T}$ will be declared by

$$L : \mathbf{S} \to \mathbf{T}$$

If $L : \mathbf{S} \to \mathbf{T}$ and $M : \mathbf{T} \to \mathbf{U}$, then the composition of these functions is denoted by a little circle

$$M \circ L : \mathbf{S} \to \mathbf{U}$$

It is defined by the formula

$$(M \circ L)(X) = M(L(X)), \qquad \text{for all } X \text{ in } \mathbf{S}$$

The identity function on **S** maps every element of **S** to itself

$$\begin{aligned} id_{\mathbf{S}} &: \mathbf{S} \to \mathbf{S} \\ id_{\mathbf{S}}(X) &= X, \qquad \text{for all } X \text{ in } \mathbf{S} \end{aligned}$$

The image of a function is the set of values it can actually take

$$image(L) =_{df} \{L(X) \mid X \in \mathbf{S}\}$$

A mathematical theory is characterised not only by the set of values which form its subject matter, but also by the collection of operators that may be applied to these values. The set of symbols chosen for the operators constitute the *signature* of the theory, usually denoted Σ. Different theories often share the same symbols from Σ, and formulae using shared symbols will look the same in both theories. It is very important that a link between these theories should map a formula in one theory to the formula in the other that has the same appearance. Such a link is called a *homomorphism*, or more precisely a *Σ-homomorphism*, where Σ is the set of shared operator symbols. In theories of programming, formulae containing the least fixed point of functions expressible in the theory should also be translated without change.

In Section 4.1 we concentrate on the simple and familiar case when the predicates of one of the two theories form a subset of those of the other ($\mathbf{T} \subseteq \mathbf{S}$, say). The link function is now an *endofunction*, mapping **S** to a subset of itself, and the predicates of **T** are just those in its image. Important properties of an endofunction are monotonicity and idempotence and weakening, and these may occur in interesting combinations. For subset theories, it is important that the result of each operation of the shared signature Σ should be a member of the subset whenever all its parameters are. A link that guarantees this is called a Σ-closure. Reasoning about closure can be generalised to a parameterised family of subsets; their closure properties define a type system, of the kind that is used for checking and optimisation by compilers for many programming languages.

The more general case of a link is a function that maps between disjoint domains. Since the domains are usually lattices, monotonicity is still an important property, and so is distribution through conjunction or disjunction over sets of various sizes, finite, infinite or empty. If a function distributes through arbitrary disjunctions, it is called a *Galois connection*, as defined and described in Section 4.2; its properties are analogous to those of the subset relation, even though its image is disjoint from its domain.

The *weakest prespecification* is an interesting generalisation of the weakest precondition (Definition 2.8.4), in which the argument is an arbitrary predicate, not restricted to just a condition. It contributes to the process of top-down program decomposition by answering the question:

Given a design Q and a specification S, what is the weakest specification X of a program to be executed before Q such that $S \sqsubseteq X; Q$?

It is an excellent example of a Galois connection, and it shares many properties with more familiar examples, such as approximate division of natural numbers and implication in logic (either classical or intuitionistic). These are the topics of Section 4.3.

The final section of the chapter deals with the case of a Galois connection that can be defined by means of a single predicate within the theory itself. This is known as a *simulation*; its purpose is to map abstract mathematical concepts onto the values which represent them concretely in the store of a computer. The simulation then provides an automatic way of transforming an abstract mathematical algorithm onto the corresponding concrete computer program. An important example is the simulation that maps the symbolic variables of a programming language to the machine addresses of the storage locations which hold their values in a computer at run time.

4.1 Subset theories

If the theory **T** is a subset of **S**, there is always a very simple link from **T** to **S**. It is just the identity function, whose domain has been restricted to **T**.

$$R : \mathbf{T} \rightarrow \mathbf{S} \quad =_{df} \quad (\lambda X : X \in \mathbf{T} \bullet X)$$

A more interesting link is to be sought in the other direction, from the more general expressive theory to the more restricted. It is a function

$$L : \mathbf{S} \rightarrow \mathbf{S}$$

from **S** to **S** itself (an endofunction), which ranges over *all* the members of **T**

$$\mathbf{T} \quad = \quad image(L)$$

This equation can be used to *define* the subset theory **T** from any chosen endofunction L. A nested sequence of subsets can be derived by composing a sequence of endofunctions, in the desired order.

Examples 4.1.1 (Endofunction)

Let P, Q, J and K be predicates, and let **S** be the set of predicates of a theory.

name	mapping
$id_{\mathbf{S}}$	$\lambda X : X \in \mathbf{S} \bullet X$
or_P	$\lambda X : X \in \mathbf{S} \bullet (P \vee X)$
and_Q	$\lambda X : X \in \mathbf{S} \bullet (Q \wedge X)$
$or_P \circ and_Q$	$\lambda X : X \in \mathbf{S} \bullet (P \vee (X \wedge Q))$
pre_J	$\lambda X : X \in \mathbf{S} \bullet (J; X)$
ass_e	$\lambda X : X \in \mathbf{S} \bullet (v := e\,;\, X)$
$post_K$	$\lambda X : X \in \mathbf{S} \bullet (X; K)$
imp_P	$\lambda X : X \in \mathbf{S} \bullet (P \Rightarrow X)$

□

A common property of a link, shared by all these examples, is *monotonicity*. But some important links do not possess it. In particular, a function which links a lattice of healthy predicates to a theory of feasible predicates has to map the miraculous predicate **false** to the feasible but useless predicate **true**. Such a function cannot be monotonic (unless it trivially maps everything to **true**). In this section we will *not* assume that endofunctions are monotonic; we will explore their other useful properties, particularly *weakening* and *idempotence*.

Definition 4.1.2 (Weakening and strengthening)

A function $L : \mathbf{S} \to \mathbf{S}$ is *weakening* if

$$L(X) \;\sqsubseteq\; X, \qquad \text{for all } X \in \mathbf{S}$$

The definition of *strengthening* is similar, using the inequality $\sqsupseteq$. □

Definition 4.1.3 (Idempotence)

A function $L : \mathbf{S} \to \mathbf{S}$ is *idempotent* if

$$L \circ L \;=\; L$$ □

Examples 4.1.4 (Weakening)

(1) or_P is weakening and and_Q is strengthening.

(2) If L and M are weakening, so is $L \circ M$. □

Examples 4.1.5 (Idempotent)

The following are idempotents, subject to the stated proviso, which we will assume to hold in future examples.

name	proviso
or_P	
and_Q	
pre_J	$J;J = J$
$post_K$	$K;K = K$
$or_P \circ and_Q$	
$pre_J \circ or_P$	$J;J = J$
$post_K \circ or_P$	$K;K = K$

□

In general, the composition of idempotents is not idempotent.

Counterexample 4.1.6

Let $J = (v < 0)$ and $Q = (v \geq 0)$. Then

$$\begin{aligned} & pre_J(and_Q(\mathbf{true})) \\ = \;& (v < 0) \\ \neq \;& \mathbf{false} \\ = \;& pre_J(and_Q(pre_J(and_Q(\mathbf{true})))) \end{aligned}$$

□

However, idempotence of the composition is assured in certain common cases.

Theorem 4.1.7 (Commuting idempotents)

If L and M are idempotents satisfying $L \circ M = M \circ L$, then $L \circ M$ is an idempotent. □

Examples 4.1.8

(1) pre_J commutes with $post_K$.

(2) or_P commutes with $post_K$ **iff** $P = P;K$. □

If L is an idempotent, its image is actually the same as its set of fixed points, because trivially

$$(\exists Y \bullet X = L(Y)) \quad \textbf{iff} \quad X = L(X)$$

If L is monotonic as well as idempotent, and $\mathbf{S}$ is a complete lattice, then its image $\mathbf{T}$ will also be a complete lattice. But take care: the join and meet operations in $\mathbf{T}$ will be different from those in $\mathbf{S}$ (see Example 4.1.9 and Figure 4.1.10). We will distinguish them notationally by subscripts ($\sqcap_\mathbf{S}$, $\sqcup_\mathbf{T}$), and we will similarly distinguish the fixed point operators $\mu_\mathbf{S}$ and $\mu_\mathbf{T}$. An even better way to avoid confusion is to discover the common conditions under which these distinctions are unnecessary, because the lattice operators have the same meaning in both the subset and the superset.

Example 4.1.9 ($\sqcap_\mathbf{S} \neq \sqcap_\mathbf{T}$)

Let $\mathbf{S} =_{df} \{\bot, x, y, z, \top\}$ and $\mathbf{T} =_{df} \{\bot, x, y, \top\}$. Define $L : \mathbf{S} \to \mathbf{T}$ by

$$L(z) =_{df} \bot$$

$$L(w) =_{df} w \qquad \text{if } w \neq z$$

Then (see Figure 4.1.10)

$$x \sqcap_\mathbf{T} y = \bot \neq z = x \sqcap_\mathbf{S} y$$ □

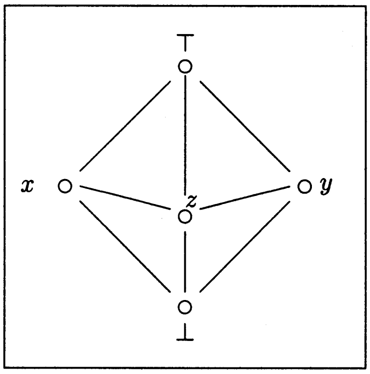

Figure 4.1.10 Not a sublattice

Definition 4.1.11 (Link and retract)

For convenience, we will reserve the term *link* for a function that is both weakening and idempotent; a *retract* is defined as a link that is monotonic as well. □

Examples 4.1.12 (retract)

(1) If $K \sqsubseteq id_{\mathbf{S}}$ then $post_K$ is a retract.

(2) If L and M are retracts, so is $L \circ M$. □

In general, a weaker theory **T** is less able to describe things accurately than a theory **S** that contains more predicates. But for any predicate P in **S**, we can single out those members X of **T** that *approximate to P*, in the sense of describing the same thing less accurately ($X \sqsubseteq P$). The *best* approximation to P is the strongest limit of this set.

Definition 4.1.13 (Approximation)

$$approx^{\mathbf{S}}_{\mathbf{T}}(P) \ =_{df} \ \bigsqcup_{\mathbf{S}} \{ X : \mathbf{T} \mid X \sqsubseteq P \} \qquad \square$$

Although in general the limit $\bigsqcup_{\mathbf{S}}$ will not be a member of **T**, we will be most interested in the cases when it is, as described in the following theorem.

Theorem 4.1.14

If $\mathbf{T} \subseteq \mathbf{S}$ and $\bigsqcup_{\mathbf{T}} = \bigsqcup_{\mathbf{S}}$, then $approx^{\mathbf{S}}_{\mathbf{T}}$ is a retract. □

Corollary $approx^{\mathbf{S}}_{\mathbf{T}}$ is the only retract linking between **S** and **T**. □

The nature of the weakest fixed point in a subset theory is explored in the following theorem, which gives the necessary assurance that the weakest fixed point exists and belongs to the subset. This is particularly important in Case (2), when L is not monotonic and the subset theory T may not even be a lattice.

Theorem 4.1.15 (Links and recursion)

Let $F : \mathbf{S} \to \mathbf{S}$ be monotonic, and $F(\mathbf{T}) \subseteq \mathbf{T}$.

Let $L : \mathbf{S} \to \mathbf{T}$ satisfy $L \circ F \sqsupseteq F \circ L$.

(1) If L is a monotonic idempotent, then $L(\mu_{\mathbf{S}} F) = \mu_{\mathbf{T}} F$

(2) If L is a link, then $L(\mu_{\mathbf{S}} F) = \mu_{\mathbf{T}} F = \mu_{\mathbf{S}} F$

Proof of (1)

$$\begin{array}{lll} & \mu_{\mathbf{T}} F = F(\mu_{\mathbf{T}} F) & \{\text{weakest fixed point}\} \\ \Rightarrow & \mu_{\mathbf{T}} F \sqsupseteq \mu_{\mathbf{S}} F & \{L \text{ is monotonic}\} \\ \Rightarrow & L(\mu_{\mathbf{T}} F) \sqsupseteq L(\mu_{\mathbf{S}} F) & \{L \text{ is idempotent}\} \\ \Rightarrow & \mu_{\mathbf{T}} F \sqsupseteq L(\mu_{\mathbf{S}} F) & \end{array}$$

$$\begin{array}{lll} & L(\mu_{\mathbf{S}} F) = L(F(\mu_{\mathbf{S}} F)) & \{L \circ F \sqsupseteq F \circ L\} \\ \Rightarrow & L(\mu_{\mathbf{S}} F) \sqsupseteq F(L(\mu_{\mathbf{S}} F)) & \{L(\mu_{\mathbf{S}} F) \in \mathbf{T}, \text{ weakest fixed point}\} \\ \Rightarrow & L(\mu_{\mathbf{S}} F) \sqsupseteq \mu_{\mathbf{T}} F & \end{array}$$

$$\begin{array}{lll}
(2) & \mu_{\mathbf{S}}F \;=\; F(\mu_{\mathbf{S}}F) & \\
\Rightarrow & L(\mu_{\mathbf{S}}F) \;=\; L(F(\mu_{\mathbf{S}}F)) & \{L \circ F \sqsupseteq F \circ L\} \\
\Rightarrow & L(\mu_{\mathbf{S}}F) \;\sqsupseteq\; F(L(\mu_{\mathbf{S}}F)) & \{\text{weakest fixed point}\} \\
\Rightarrow & L(\mu_{\mathbf{S}}F) \sqsupseteq \mu_{\mathbf{S}}F & \{L(X) \sqsubseteq X\} \\
\Rightarrow & L(\mu_{\mathbf{S}}F) = \mu_{\mathbf{S}}F &
\end{array}$$

This shows that $\mu_{\mathbf{S}}F$ satisfies the defining property for membership of $\mathbf{T}$. Since $\mathbf{T} \subseteq \mathbf{S}$, it is the weakest fixed point in $\mathbf{T}$ as well

$$\mu_{\mathbf{S}}F = \mu_{\mathbf{T}}F \qquad \square$$

Corollary Let L be strengthening, and satisfy $L \circ F \;\sqsubseteq\; F \circ L$; then

$$\nu_{\mathbf{S}}F \;=\; \nu_{\mathbf{T}}F \qquad \square$$

That concludes our study of the links between a theory and a subset theory, insofar as each theory is represented purely by its set of predicates. We turn next to an equally important aspect, namely the set of operators which are introduced in the theory to build up complex formulae, predicates and programs from their primitive components. Again, the subset theory will often select only a subset Σ of the operators of the larger theory, and we will assume that all the operators are monotonic.

The main reason for omitting an operator from a subset theory is that when applied to operands in the subset $\mathbf{T}$, it does not necessarily give an answer within that subset. So let us suppose that Σ excludes all such unconstrained operators. Then $\mathbf{T}$ is said to be Σ-*closed*, and the function L that defines $\mathbf{T}$ is a Σ-closure. It is sometimes adequate that the closure be merely approximate.

Definition 4.1.16 (Σ-closure with strengthening or weakening)

Let $F : \mathbf{S} \to \mathbf{S}$ be a monotonic function. L is an F-closure if

$$L \circ F \;=\; F \circ L$$

An analogous definition is given for $F_{\sqsubseteq}$-closure or $F_{\sqsupseteq}$-closure, using inequations

$$L \circ F \;\sqsubseteq\; F \circ L \qquad \text{or} \qquad L \circ F \;\sqsupseteq\; F \circ L \qquad \square$$

The definition of closure is adapted to apply to operators of less or greater arity than one. For example, if K is a constant, L is a $K_{\sqsupseteq}$-closure if

$$L(K) \;\sqsupseteq\; K\ .$$

If G has two arguments, the condition is

$$L(G(X, Y)) \;\sqsupseteq\; G(L(X), L(Y)), \qquad \text{for all } X, Y \in \mathbf{S}$$

The definition extends to all the operators of the signature.

Definition 4.1.17 (Σ-link)

A link L is called a Σ-link if it is an F-closure for all F in Σ, and similarly for a $\Sigma_{\sqsubseteq}$-link and a $\Sigma_{\sqsupseteq}$-link. $\square$

Examples 4.1.18 (Σ-closure)

The table gives the operators in Σ for which the function is a closure or an approximate closure.

name	closure	$_{\sqsubseteq}$closure	$_{\sqsupseteq}$closure	proviso
or_P	$\sqcap,\ \lhd b \rhd$		$;$	$P;P = P$
and_Q	$\sqcap,\ \lhd b \rhd$	$;$		$Q;Q = Q$
pre_J	$\sqcap$		$;$	$J \sqsubseteq id_{\mathbf{S}}$
$post_K$	$\sqcap,\ \lhd b \rhd$		$;$	$K \sqsubseteq id_{\mathbf{S}}$

$\square$

Theorem 4.1.19

If L is a $\Sigma_{\sqsupseteq}$-link, then the set *image*(L) is closed under all functions in Σ.

Proof For any $X \in image(L)$ and any function F in Σ

$$\begin{array}{ll} & L(F(X)) & \qquad \{L \circ F \sqsupseteq F \circ L\} \\ \sqsupseteq & F(L(X)) & \qquad \{L \text{ is idempotent}\} \\ = & F(X) & \qquad \{L \text{ is weakening}\} \\ \sqsupseteq & L(F(X)) & \qquad \square \end{array}$$

The important property of a Σ-link is that it applies not just to the finite set of operators in Σ but also to the infinite set of all formulae definable by means of these operators: any program containing only constants and operators of Σ will always denote a predicate in the subset theory $\mathbf{T}$. But what if the program contains recursion, as most programs do? Fortunately, Theorem 4.1.15 shows that the closure property is strong enough to ensure that recursion preserves closure in $\mathbf{T}$.

A successful definition of a subset theory $\mathbf{T}$ requires a proof that $\mathbf{T}$ is Σ-closed for the desired operators in Σ. The proof involves a collection of theorems of the form

$$\text{If } X \in \mathbf{T} \text{ and } Y \in \mathbf{T} \ \text{ then } \ (X;Y) \in \mathbf{T}$$

These theorems are often presented in the notations of type theory, for example

$$X : \mathbf{T}, Y : \mathbf{T} \vdash (X;Y) : \mathbf{T}$$

Here, *membership* is represented by colon, and *and* by comma and the consequent is separated from the antecedents by $\vdash$. Nearly all the theorems of Exercise 3.2.6(1) can be cast in this form.

It is often useful to single out not just a single subset of a theory, but rather a whole family of subsets, indexed by a set of parameters that correspond to important properties of the subset. An operator applied to an operand in one member of the family may give a result which is in another member of the family, characterised by different parameters. For example,

$$\text{If } X = \mathit{and}_P(X) \text{ and } Y = \mathit{and}_Q(Y) \text{ then } (X;Y) = \mathit{and}_{P;Q}(X;Y)$$

In the notation of type theory, this could be rewritten

$$X : \mathit{image}(\mathit{and}_P),\ Y : \mathit{image}(\mathit{and}_Q) \vdash (X;Y) : \mathit{image}(\mathit{and}_{P;Q})$$

A collection of laws of this kind may contain enough laws to determine a type for every term in the language; it is then known as a *type system* or a *type theory.* A useful type system will permit an automatic association of a type with every term in the language – often a unique type or a minimal type of some kind. But in this book we will not be further concerned with the interesting and important techniques of automatic type inference.

As an example we will present again the theory of preconditions and postconditions (Section 2.8) in the form of a type theory. We will adopt the notation

$$X : p \to r \quad \text{instead of} \quad X \sqsupseteq (p \Rightarrow r') \quad \text{or} \quad X = (p \Rightarrow r') \wedge X$$

Theorem 4.1.20

(1) $(\mathit{\Pi} : p \to p)$, for all conditions p

(2) $X : p \to q,\ Y : q \to r \vdash (X;Y) : p \to r$

(3) $X : b \wedge p \to r,\ Y : \neg b \wedge p \to r \vdash (X \lhd b \rhd Y) : p \to r$

(4) $X : p \to r,\ Y : q \to s \vdash (X \sqcap Y) : p \wedge q \to r \vee s$

(5) If $X : p \to r$ implies $F(X) : p \to r$ for all X, then $\nu F : p \to r$

Proof of (5) From the assumption it follows that the sublattice $\mathbf{T}$ associated with the idempotent $\mathit{and}_{p \Rightarrow r'}$ is an F-closure. From the corollary of Theorem 4.1.15 we conclude

$$\nu F = \nu_{\mathbf{T}} F = \mathit{and}_{p \Rightarrow r'}(\nu_{\mathbf{T}} F) \qquad \square$$

These laws are very similar in appearance to the laws which define the algebraic concept of a *category*. The conditions p, r are the *objects* of the category, and the *arrows* are triples (p, X, r) such that $(X : p \dashrightarrow r)$. The first two laws give the basic properties of categorical composition. They are supplemented in category theory by the associative law for composition and by unit laws for Π, which are also true for programs. In summary, programs are arrows of a category whose objects are conditions. But no further knowledge of category theory will be used in this book.

A frequent goal in the definition of a subset theory is to ensure the validity on that subset of certain additional algebraic laws. For example, the desired law may be of the form

$$X \;=\; L(X)$$

Now if L happens to be idempotent, the goal is very simply achieved: just take the link L itself and define

$$\mathbf{T} =_{df} image(L)$$

As an additional advantage, it is frequently found that other useful laws become true in $\mathbf{T}$, as shown by the examples taken from Chapter 3.

Examples 4.1.21 (Healthiness conditions)

(1) These examples are taken from Section 3.2. Define the following retracts

$$\begin{array}{lll} \mathbf{H1} & =_{df} & or_{\neg ok} \\ \mathbf{H2} & =_{df} & post_K \quad \text{where} \quad K =_{df} (ok \Rightarrow ok') \wedge (v' = v) \\ \mathbf{H3} & =_{df} & post_{\Pi} \quad \text{where} \quad \Pi =_{df} ok \Rightarrow (ok' \wedge v' = v) \end{array}$$

Then

$X \in image(\mathbf{H1})$ **iff** X satisfies the healthiness condition **H1**

$X \in image(\mathbf{H2})$ **iff** X satisfies the healthiness condition **H2**

$X \in image(\mathbf{H3})$ **iff** X satisfies the healthiness condition **H3**

(2) Let $\mathbf{H4}(X) =_{df} ((X; \mathbf{true}) \Rightarrow X)$, where X is a design. Then

(2a) the mapping **H4** is a $\Sigma_{\sqsupseteq}$-link for $\Sigma = \{\sqcap, ;, \lhd b \rhd\}$, and

(2b) $X \in image(\mathbf{H4})$ **iff** X satisfies the healthiness condition **H4**. □

Exercises 4.1.22

Suppose that the following laws hold for all $X \in \{J, T, B\}$

L1 $X = X \vee T$ (new top)

L2 $X = X \wedge B$ (new bottom)

L3 $J;X = X = X;J$ (new unit)

L4 $X;X = X$ (idemp)

Prove that

(1) P satisfies the first three laws **iff** it is a fixed point of the monotonic mapping

$$L(X) =_{df} J;(T \vee B \wedge X);J$$

(2) If $B;T = B$, then all fixed points P of L satisfy the left zero law

$$B;P = B$$

□

4.2 Galois connections

One of the strongest familiar properties of a function L is that it is a *bijection.* This means that there exists a function L^{-1} which exactly reverses the effect of L, and the same reversal occurs on applying the functions in the opposite order

$$L^{-1}(L(X)) = X \quad \text{and} \quad L(L^{-1}(Y)) = Y$$

This property can be expressed more abstractly as

$$L \circ L^{-1} = id_{\mathbf{T}} \quad \text{and} \quad L^{-1} \circ L = id_{\mathbf{S}}$$

Two theories which are connected by a bijection have equal expressive power. They may have different alphabets and they may be presented in different styles; they may be implemented with differing efficiency or in different technologies. But everything that can be expressed in one theory can be expressed just as precisely in the other. In this mathematical sense, a bijection is the strongest possible kind of a function to serve as a link between the predicates of two theories. The only trouble is that theories can be *too* similar, because there is *no* interesting mathematical distinction between them.

A mathematically more interesting case is when one theory is genuinely richer than another; it contains features that cannot be exactly mapped into the weaker theory. As a result of the weakness, the function L which maps each predicate of the stronger theory to the weaker one can only give a certain *best approximation* of the same meaning in the weaker theory. So there cannot be an exact inverse for L. Nevertheless, there often exists a function R, which maps in the opposite direction (from the weaker to the stronger), and as far as possible undoes the effect of L. But there is some unavoidable weakening, so the equality which holds for a bijection has to be replaced by an inequation

$$X \sqsupseteq R(L(X))$$

However, when the two functions are applied in the opposite order, they may actually strengthen the predicate Y in the weaker theory

$$L(R(Y)) \sqsupseteq Y$$

It is this second law that ensures that the weakening caused by the function L is as small as possible. Such a pair of functions $(L,\ R)$ is known as a *Galois connection.*

Definition 4.2.1 (Galois connection)

Let $\mathbf{S}$ and $\mathbf{T}$ both be complete lattices. Let L be a function from $\mathbf{S}$ to $\mathbf{T}$, and let R be a function from $\mathbf{T}$ to $\mathbf{S}$. The pair $(L,\ R)$ is a *Galois connection* if for all $X \in \mathbf{S}$ and $Y \in \mathbf{T}$

$$L(X) \sqsupseteq Y \quad \textbf{iff} \quad X \sqsupseteq R(Y)$$

R is called a *weak inverse* of L, and L is called a *strong inverse* of R. (Sometimes they are called left adjoint (L) and right adjoint (R).) The restriction to complete lattices is not necessary; the definition applies to any partial order. □

Example 4.2.2 (Conjunction and implication)

Let $and_P(X) =_{df} P \wedge X$ and $imp_P(Y) =_{df} (P \Rightarrow Y)$. It is a simple fact of the predicate calculus that

$$(P \wedge X) \sqsupseteq Y \quad \textbf{iff} \quad X \sqsupseteq (P \Rightarrow Y)$$ □

In Section 1.4, this example illustrated a top-down development technique for designing a component X to fit into an assembly $L(X)$, with the goal that it should meet a specification Y. The weakest allowable specification of the component to meet that goal is calculated as $R(Y)$. The general definition of a Galois connection extends the technique of top-down development to the case when the specification Y is actually expressed in a different theory from the description of the component X. For example, the two theories may have different alphabets, as described in Section 1.6.

The following lemma gives an alternative definition for a Galois connection, which matches the alternative definition of a bijection.

Lemma 4.2.3

Let L and R be monotonic mappings. (L, R) is a Galois connection **iff**

$$(L \circ R) \sqsupseteq id_{\mathbf{T}} \text{ and } id_{\mathbf{S}} \sqsupseteq (R \circ L) \qquad \square$$

Exercise 4.2.4

Prove that if (L, R) is a Galois connection, then L and R are monotonic. □

This exercise shows that not all bijections are Galois connections. For example, negation is its own exact inverse, but it is far from being monotonic.

Theorem 4.2.5 (Composition of Galois connections)

If (L_1, R_1) and (L_2, R_2) are Galois connections, then their composition

$$(L_2 \circ L_1,\ R_1 \circ R_2)$$

is alao a Galois connection. □

As in the case of a bijection, a Galois connection is uniquely determined by either of its two components separately.

Theorem 4.2.6 (Uniqueness of adjoint)

If (L_1, R_1) and (L_2, R_2) both are Galois connections, then

$$(L_1 = L_2) \quad \textbf{iff} \quad (R_1 = R_2)$$

Proof First we show that $(L_1 = L_2) \Rightarrow (R_1 = R_2)$.

$$\begin{array}{lll} & X \sqsupseteq R_1(Y) & \{\text{Def. 4.2.1}\} \\ \equiv & L_1(X) \sqsupseteq Y & \{L_1 = L_2\} \\ \equiv & L2(X) \sqsupseteq Y & \{\text{Def. 4.2.1}\} \\ \equiv & X \sqsupseteq R_2(Y) & \end{array}$$

which implies that $R_1 = R_2$. In a dual way, replacing $\sqsupseteq$ with $\sqsubseteq$, we can show $(R_1 = R_2) \Rightarrow (L_1 = L_2)$. □

A Galois connection can be drawn as a picture, with **T** and **S** represented by plane figures and the functions L and R by arrows in one direction or the other. Figure 4.2.7 shows how L maps the whole of **S** to a subset in the upper areas of **T**, and R dually selects a subset towards the bottom of **S**. The connection between these two subsets is a bijection, showing that each subspace contains an exact image of the corresponding part of the other. More formally, if X is in the image of L then $L(R(X)) = X$, and if Y is in the image of R the $R(L(Y)) = Y$.

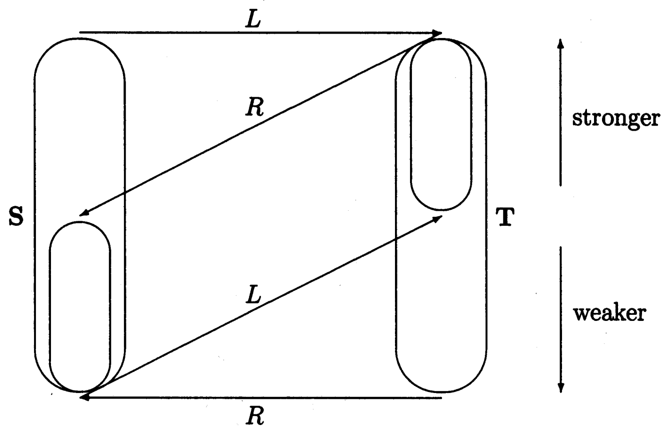

Figure 4.2.7 Galois connection

An interesting special case is when the image of L is the whole of $\mathbf{T}$, that is L is a surjection. This case is captured by a definition in terms of its most important property.

Definition 4.2.8 (Retract and coretract)

A Galois connection (L, R) is a *retract* if

$$L \circ R \ = \ id_{\mathbf{T}}$$

It is a *coretract* if $R \circ L \ = \ id_{\mathbf{S}}$. □

A common special case of a retract is when the predicates of the weaker theory $\mathbf{T}$ are selected as a subset of the stronger theory $\mathbf{S}$. In this case, the weak inverse R is just the identity function on $\mathbf{S}$, but restricted to $\mathbf{T}$.

$$R(X) \ = \ X, \qquad \text{for all } X \text{ in } \mathbf{T}$$

Furthermore, the left inverse L is exactly the approximation $approx^{\mathbf{S}}_{\mathbf{T}}$ (Definition 4.1.13). More generally, for any Galois connection (L, R), the function $R \circ L$ is a retract on $\mathbf{S}$. This shows the high degree of similarity between Galois connections and subsets.

In Section 2.5 we gave definitions of the conjunctive and disjunctive properties of functions. There are a number of useful weaker versions of these properties, depending on the size of the set over which the lower or upper bound is taken.

Definition 4.2.9 (Disjunctivity and conjunctivity)

Let **U** (not necessarily a lattice) be a subset of **S**. Consider the distributive laws

$$(1)\quad L(\bigsqcup \mathbf{U}) \;=\; \bigsqcup\{L(X) \mid X \in \mathbf{U}\}$$
$$(2)\quad L(\bigsqcap \mathbf{U}) \;=\; \bigsqcap\{L(X) \mid X \in \mathbf{U}\}$$

where $\bigsqcup$ and $\bigsqcap$ stand for the *least upper bound* and *greatest lower bound* operators respectively. L is said to be

$\bigsqcup^{\infty}$ (universally conjunctive) if the first law is true for all subsets **U**.

$\bigsqcup^{+}$ (positively conjunctive) if the first law is true for all non-empty **U**.

$\bigsqcup^{2}$ (conjunctive) if the first law is true for all finite and non-empty **U**.

$\bigsqcup^{0}$ (strict on bottom) if the first law is true for empty **U**.

Similarly, using the second distributive law as a criterion, we can define a function to be $\bigsqcap^{\infty}$ (universally disjunctive), $\bigsqcap^{+}$ (positively disjunctive), $\bigsqcap^{2}$ (disjunctive) and $\bigsqcap^{0}$ (strict on top). □

Examples 4.2.10 (Disjunctivity and conjunctivity)

name	disjunctivity	conjunctivity
or_P	$\bigsqcap^{+}$	$\bigsqcup^{\infty}$
and_Q	$\bigsqcap^{\infty}$	$\bigsqcup^{+}$
pre_J	$\bigsqcap^{\infty}$	
$post_K$	$\bigsqcap^{\infty}$	

Distributivity gives us an exact criterion for a Galois connection.

Theorem 4.2.11 (Distributivity and Galois connections)

(1) The weak inverse of L exists **iff** L is universally disjunctive.

(2) The strong inverse of R exists **iff** R is universally conjunctive.

Proof Assume that L has a weak inverse R. For any subset **U** of **S** and any $Y \in \mathbf{T}$

$$
\begin{aligned}
& L(\bigsqcap \mathbf{U}) \sqsupseteq Y && \{\text{Def. 4.2.1}\}\\
\equiv\ & \bigsqcap \mathbf{U} \sqsupseteq R(Y) && \{\text{2.5}\mathbf{L1}\}\\
\equiv\ & \forall X \in \mathbf{U} \bullet (X \sqsupseteq R(Y)) && \{\text{Def. 4.2.1}\}\\
\equiv\ & \forall X \in \mathbf{U} \bullet (L(X) \sqsupseteq Y) && \{\text{2.5}\mathbf{L1}\}\\
\equiv\ & (\bigsqcap\{L(X) \mid X \in \mathbf{U}\}) \sqsupseteq Y
\end{aligned}
$$

So L is universally disjunctive.

Suppose that L is $\sqcap^\infty$. Let $R(Y) =_{df} \sqcap\{P \mid L(P) \sqsupseteq Y\}$. This is the weakest predicate in **S** whose transform by L meets specification Y in **T**.

$$\begin{array}{llr}
 & L(X) \sqsupseteq Y & \{\text{set theory}\} \\
\equiv & X \in \{P \mid L(P) \sqsupseteq Y\} & \{\text{def of } R\} \\
\Rightarrow & X \sqsupseteq R(Y) & \{L \text{ is mono}\} \\
\Rightarrow & L(X) \sqsupseteq L(R(Y)) & \{L \text{ is } \sqcap^\infty\} \\
\equiv & L(X) \sqsupseteq \sqcap\{L(P) \mid L(P) \sqsupseteq Y\} & \{2.5\mathbf{L1}\} \\
\Rightarrow & L(X) \sqsupseteq Y &
\end{array}$$

The conclusion (2) can be proved in a similar way. □

Fixed points in Galois-connected theories are related in much the same way as in a subset theory: again, the theorems depend on commutativity between the link and the monotonic function in question.

Theorem 4.2.12 (μ-fusion [121])

Let F and G be monotonic functions, and let (L, R) be a Galois connection.

(1) If $R \circ F \sqsubseteq G \circ R$, then $R(\mu F) \sqsubseteq \mu G$

(2) If $R \circ F = G \circ R$, then $R(\mu F) = \mu G$

Proof of (1)

$$\begin{array}{llr}
 & (R \circ F) \sqsubseteq (G \circ R) & \{L \text{ is monotonic}\} \\
\Rightarrow & (L \circ R \circ F \circ L) \sqsubseteq (L \circ G \circ R \circ L) & \{\text{Lemma 4.2.3}\} \\
\Rightarrow & (F \circ L) \sqsubseteq (L \circ G) & \{\text{def of } \sqsubseteq\} \\
\Rightarrow & F(L(\mu G)) \sqsubseteq L(G(\mu G)) & \{\text{fixed point}\} \\
\equiv & F(L(\mu G)) \sqsubseteq L(\mu G) & \{\text{weakest fixed point}\} \\
\Rightarrow & \mu F \sqsubseteq L(\mu G) & \{\text{Def. 4.2.1}\} \\
\equiv & R(\mu F) \sqsubseteq \mu G &
\end{array}$$

(2)

$$\begin{array}{llr}
 & (R \circ F) = (G \circ R) & \{\text{Exercise 2.6.4(2)}\} \\
\Rightarrow & (R \circ F) = (G \circ R) \wedge (R(\mu F) \sqsupseteq \mu G) & \{\text{Conclusion (1)}\} \\
\Rightarrow & R(\mu F) = \mu G &
\end{array}$$

□

It is time now to turn from the predicates of the two theories to their respective signatures $\Sigma_{\mathbf{S}}$ and $\Sigma_{\mathbf{T}}$. It is very common that these signatures should contain many or all of their symbols in common, and we will confine attention to the symbols that they share

$$\Sigma = \Sigma_{\mathbf{S}} \cap \Sigma_{\mathbf{T}}$$

In the case of a subset theory, it is reasonable to insist that the same symbol has the same meaning in both theories, and the concept of Σ-closure (Definition 4.1.16) was defined to give the necessary assurance. But in the case of a Galois connection $\mathbf{S}$ and $\mathbf{T}$ are typically disjoint sets, and no operator can have the same meaning on both of them. Nevertheless, it is possible to use the same symbol with two different meanings, and without too much confusion.

The confusion is avoided in the study of algebra by defining the appropriate properties of the function that connects the theories. In particular, it should commute with all operators in Σ.

Definition 4.2.13 (Σ-morphism)

A function $L : \mathbf{S} \to \mathbf{T}$ is an F-morphism if

$$L \circ F_{\mathbf{S}} \ = \ F_{\mathbf{T}} \circ L$$

where the subscripts emphasise (for the last time) that the operators have disjoint meanings in $\mathbf{S}$ and $\mathbf{T}$.

L is an $F_{\sqsubseteq}$-morphism if the equality above is replaced by $\sqsubseteq$, and an $F_{\sqsupseteq}$-morphism is defined similarly. The definition is easily adapted to operators of any arity. A Σ-morphism is an F-morphism for all F in Σ. □

Example 4.2.14

Let $(L,\ R)$ be a Galois connection.
Then L is a $\Sigma_{\sqsupseteq}$-morphism **iff** R is a $\Sigma_{\sqsubseteq}$-morphism. □

Exercises 4.2.15

In the following, assume that (L, R) and (L_i, R_i), for $i \ = \ 1, 2$, are Galois connections for appropriate domains and ranges. Prove the claims

(1) $L \circ R \circ L \ = \ L$ and $R \circ L \circ R \ = \ R$.

(2) Define $(L_1 \sqcap L_2)(X) =_{df} L_1(X) \sqcap L_2(X)$, and $(R_1 \sqcup R_2)(X)$ similarly. Then $(L_1 \sqcap L_2,\ R_1 \sqcup R_2)$ is a Galois connection.

(3) Define the iterates

$$\begin{aligned} L^{\diamond}(X) &=_{df} (\nu Y \bullet X \sqcap L(Y)) \\ R^{\square}(X) &=_{df} (\mu Y \bullet X \sqcup R(Y)) \end{aligned}$$

Then $L^{\diamond}$ is a retract, and $R^{\square}$ a coretract.

(4) Burghard von Karger has shown that a Galois connection $(L,\ R)$ forms the basis for temporal logic [118], with the definitions [108]

$$\begin{array}{llll}
\oplus P & =_{df} & L(P) & \text{forward step (next)} \\
\Diamond\!\!\!\!+\, P & =_{df} & L^{\Diamond}(P) & \text{in the future (sometime)} \\
\ominus P & =_{df} & \neg R(\neg P) & \text{backward step (previously)} \\
\Diamond\!\!\!\!-\, P & =_{df} & \neg R^{\Box}(\neg P) & \text{in the past (sometime)}
\end{array}$$

The duals of these give the "always" operators, often written with a box

$$\boxplus(P) \;=_{df}\; \neg\Diamond\!\!\!\!+\,(\neg P) \qquad \text{and} \qquad \boxminus(P) \;=_{df}\; \neg\Diamond\!\!\!\!-\,(\neg P)$$

With these definitions, prove the following laws of temporal logic

$$\begin{array}{rcl}
\oplus P \wedge \boxplus Q & \sqsupseteq & \oplus(P \wedge Q) \\
\oplus P \wedge Q & \sqsupseteq & \oplus(P \wedge \ominus Q)
\end{array}$$

□

4.3* Prespecification and postspecification

The treatment of Galois connections in Section 4.2 dealt with functions only of a single argument. But a function with two (or more) arguments can be regarded as a single-argument function, if the values of all the other arguments are fixed. For example, the functions pre_J and $post_K$ of Section 4.1 are derived from sequential composition by fixing the first or second of its arguments to J or K respectively. Each of these functions is universally disjunctive, and so they both have approximate inverses, which form the subject of this section.

Definition 4.3.1 (Weakest prespecification and postspecification)

The weakest prespecification of K through X is denoted by X/K and is defined as the weak inverse of $post_K$

$$(X;K) \sqsupseteq Y \quad \textbf{iff} \quad X \sqsupseteq (Y/K)$$

Because $post_K$ is $\sqcap^{\infty}$, there is no need to give an explicit formula for (Y/K); the above implicit definition characterises it uniquely. The weakest postspecification is defined similarly by

$$(J;X) \sqsupseteq Y \quad \textbf{iff} \quad X \sqsupseteq (J\backslash Y)$$

□

The weakest prespecification [92] is a generalisation of the weakest precondition (Definition 2.8.4); they are almost identical when the first argument mentions only dashed variables

$$r'/K \;=\; (K \;\mathbf{wp}\; r)'$$

The following laws are generalisations of the laws 2.8**L10** to **L13**

L1 $Q(x')/(x := e) \;=\; Q(e')$

L2 $S/(P \lhd b \rhd Q) \;=\; (S/P) \lhd b' \rhd (S/Q)$

L3 $S/(P;Q) \;=\; (S/Q)/P$

L4 $S/(P \sqcap Q) \;=\; (S/P) \sqcup (S/Q)$

L5 $(R \sqcup S)/P \;=\; (R/P) \sqcup (S/P)$

L6 $P \sqsupseteq (P;Q)/Q$

L7 $(S/Q);Q \sqsupseteq S$

L8 $(S/P);(P/Q) \sqsupseteq (S/Q)$

L9 $I\!I \sqsupseteq S/S$

The laws for postspecification are exact mirror images of the laws for prespecification, for example $(Q;P)\backslash S \;=\; P\backslash(Q\backslash S)$. This is obtained from **L3** for / by reading the equation backwards, exchanging / for \ but leaving ; unchanged. The following law requires both operators and it is its own mirror image

L10 $(Q\backslash S)/P \;=\; Q\backslash(S/P)$

We have previously (Section 1.6) drawn attention to the analogy between stepwise decomposition of program designs and factorisation of natural numbers, and the same analogy extends between weakest prespecification and approximate division (discarding the remainder). The basis of the analogy is now revealed: both of them are Galois connections.

$$x \times q \leq y \quad \textbf{iff} \quad x \leq y \div q, \qquad\qquad \text{for } x, y \geq 0 \text{ and } q > 0$$

From this single property the following laws can be deduced in exactly the same way as the laws **L3** to **L9** for /. Here, $\sqcup$ means the lesser of its two arguments and $\sqcap$ is the greater; the divisors are assumed to be non-zero.

$$\begin{array}{rcl}
r \div (p \times q) & = & (r \div q) \div p \\
r \div (p \sqcap q) & = & (r \div p) \sqcup (r \div q) \\
(r \sqcup s) \div p & = & (r \div p) \sqcup (s \div p) \\
p & = & (p \times q) \div q \\
(p \div q) \times q & \leq & p \\
(p \div q) \times (q \div r) & \leq & p \div r \\
1 & = & p \div p
\end{array}$$

Another interesting analogy is that between programming operators and logical operators, given in the following table.

$;$	$\wedge$
$/$	$\Rightarrow$
$\sqcup$	$\wedge$
$\sqcap$	$\vee$
$\sqsupseteq$	$\vdash$

The symbol $\vdash$ denotes deduction: $P \vdash Q$ means that Q can be validly deduced from P, and $P \dashv\vdash Q$ means that the reverse deduction is also valid. It is reasonably assumed that

$$X \dashv\vdash X, \qquad \text{for all } X$$

The basic Galois connection of propositional calculus is

$$(P \wedge X) \vdash Y \quad \textbf{iff} \quad X \vdash (P \Rightarrow Y)$$

The consequences of the connection yield many of the familiar proof rules of propositional calculus, including *modus ponens*

$$\begin{array}{rcl}
(q \wedge p) \Rightarrow r & \dashv\vdash & p \Rightarrow (q \Rightarrow r) \\
(p \vee q) \Rightarrow r & \dashv\vdash & (p \Rightarrow r) \wedge (q \Rightarrow r) \\
p \Rightarrow (r \wedge s) & \dashv\vdash & (p \Rightarrow r) \wedge (q \Rightarrow s) \\
p & \vdash & q \Rightarrow (p \wedge q) \\
(q \Rightarrow p) \wedge q & \vdash & p \\
(r \Rightarrow q) \wedge (q \Rightarrow p) & \vdash & r \Rightarrow p \\
q & \vdash & (q \Rightarrow p) \Rightarrow p
\end{array}$$

The Galois connection between $\wedge$ and $\Rightarrow$ is the basis of an elegant presentation of intuitionistic logic [55]. Linear logic [66] is a variety of logic that distinguishes clearly between the (multiplicative) conjunction that corresponds to ; and the (linear) conjunction that corresponds to $\sqcup$. The surprisingly wide range of application of the Galois connections of a binary operator has aroused interest in a branch of mathematics known as quantale theory [162].

In the stepwise development of a design to meet a specification S, it may be decided to split the task into two parts P and Q, which are specifications or designs for two fragments of program that will be executed sequentially. To check that this design decision is sound, a proof must be provided (preferably before the implementation) that

$$P;Q \ \sqsupseteq\ S$$

As explained in Section 1.5, the design task can be greatly simplified if only one of the designs (say Q) needs to be formulated in detail; the other one (P) should then be obtained purely by calculation from S and Q, with the guarantee that the resulting design will be correct. The calculation should give the weakest allowable specification for P, because that is the one that gives the greatest chance of easy implementation. It is the weakest design of a program to be executed *before* Q in order to achieve S. In other words, it is the weakest prespecification S/Q, and its role in stepwise design is just an intuitive restatement of its defining property (Definition 4.3.1). The weakest postspecification serves a similar purpose if the first component of the sequential composition is designed first.

The weakest prespecification of a design is a design[1]; the only danger is that it may not be a feasible design. Infeasibility occurs when there is no way that a program that ends in Q can achieve the given specification. For example,

$$(\mathbf{true} \vdash n' \text{ is odd})/(n := 2 \times n) \;=\; \top$$

In the rest of this section we introduce Conway's theory of factors [43] which explores all possible ways of implementing a specification S by sequential composition of designs. Consider law **L7**

$$(S/Q);Q \;\sqsupseteq\; S$$

(S/Q) is guaranteed to be the easiest design that will serve its intended purpose in the intended context shown above. But is Q the easiest valid design that will serve for a second component? Perhaps in knowledge of its intended context of use, it could be made weaker, and therefore easier to implement. This question is answered by the weakest postspecification, because by its definition

$$(S/Q);X \;\sqsupseteq\; S \quad \textbf{iff} \quad X \;\sqsupseteq\; (S/Q)\backslash S$$

This means that, in the given context, Q can always validly be replaced by $(S/Q)\backslash S$, which is always a weakening

$$Q \;\sqsupseteq\; (S/Q)\backslash S$$

If Q is already the weakest possible design in the circumstance, it is called a *right factor* of S [43]. A left factor has a similar definition.

Definition 4.3.2 (Right and left factor)

Q is a right factor of S if

$$Q \;=\; (S/Q)\backslash S$$

Q is a left factor of S if $Q \;=\; S/(Q\backslash S)$. □

[1]Note that this is *not* true of the weakest postspecification.

Any trick that can be played once can be played again, but not perhaps with any useful effect. From the law

$$(S/Q);((S/Q)\backslash S) \sqsupseteq S$$

we can try this time to weaken the first component (S/Q). But there is no point because in fact it is already as weak as possible, as shown by the law

L11 $(S/Q) = S/((S/Q)\backslash S)$

We explore next the set of right factors of S. This is done with the aid of a linking function which maps each Q onto the strongest right factor of S which is weaker than Q.

Definition 4.3.3 (Link to right factor)

$$L_S(Q) =_{df} (S/Q)\backslash S \qquad \square$$

Theorem 4.3.4

L_S is a retract.

Proof From **L7** and **L11** it follows that L_S is a link. From **L4** and **L5** we conclude that L_S is also monotonic. □

Theorem 4.3.5 (Right factor)

The following conclusions are equivalent.

(1) Q is a right factor of S.

(2) There is an X such that $Q = (X\backslash S)$.

(3) There is a P such that the following laws hold

(a) $X;Q \sqsupseteq S$ **iff** $X \sqsupseteq P$

(b) $P;X \sqsupseteq S$ **iff** $X \sqsupseteq Q$ □

Theorem 4.3.6 (Transitivity)

If Q is a right factor of R and R is a right factor of S, then Q is a right factor of S.

Proof From Theorem 4.3.5 it follows that there exist X and Y such that

$$Q = X\backslash R \quad \text{and} \quad R = Y\backslash S$$

It follows that

$$Q = X\backslash R = X\backslash(Y\backslash S) = (Y;X)\backslash S \qquad \square$$

4.4* Simulation

In the arithmetic of fractions and reals, reasoning about division is greatly simplified by the existence of reciprocals of non-zero numbers, and similarly in matrix arithmetic, non-singular matrices have exact inverses. Multiplication of a number d by its reciprocal u (in either order) always gives the unit of multiplication

$$d \times u \ = \ 1 \quad \text{and} \quad u \times d \ = \ 1$$

In this section we concentrate on relations that have reciprocals with respect to composition (in place of multiplication), but only in an approximate sense, defined by weakening the above two equations to inequations in opposing directions.

Definition 4.4.1 (Simulation)

Let D and U be designs satisfying

$$(D; U) \ \sqsupseteq \ \mathit{II} \quad \text{and} \quad \mathit{II} \ \sqsupseteq \ (U; D)$$

In this case, D is called a *simulation* and U is a *co-simulation*; the pair (D, U) is also called a simulation. □

Examples 4.4.2

(1) $(b^{\top}, b_{\perp})$ is a simulation among designs, because

$$(b^{\top}; b_{\perp}) \ = \ b^{\top} \ \sqsupseteq \ \mathit{II} \ \sqsupseteq \ b_{\perp} \ = \ (b_{\perp}; b^{\top})$$

(2) The design $(\mathbf{var}\, x\,;\ x := f(y);\ \mathbf{end}\, y)$ is a co-simulation. Its simulation is a non-deterministic assignment to y of an arbitrary element of $\{m \mid f(m) = x\}$, if this is non-empty, otherwise $\perp$. □

Theorem 4.4.3 (Simulation and Galois connection)

If (D, U) is a simulation then

$$(\mathit{pre}_D \circ \mathit{post}_U,\ \mathit{pre}_U \circ \mathit{post}_D)$$

is a Galois connection. □

A simulation uniquely determines its co-simulation and vice versa.

Theorem 4.4.4

If (D_1, U_1) and (D_2, U_2) are simulations then

$$(D_1 \ = \ D_2) \quad \textbf{iff} \quad (U_1 \ = \ U_2)$$ □

To take advantage of this theorem, it is necessary to have an independent test whether a predicate is a co-simulation, and the test should be simpler than formalising its partner in the other direction.

Theorem 4.4.5

U is a co-simulation **iff** $U/U \;=\; U;(\Pi/U)$.

Proof $(\Leftarrow)$ Let $D \;=\; \Pi/U$.

$$\begin{array}{ll} & \Pi \qquad\qquad \{4.3\mathbf{L9}\} \\ \sqsupseteq & U/U \qquad\qquad \{\text{assumption and def of } D\} \\ = & U;D \end{array}$$

The inequality $(D;U) \sqsupseteq \Pi$ follows directly from Definition 4.3.1.

$$\begin{array}{lll} (\Rightarrow) & & (Y;U) \sqsupseteq X \qquad\qquad \{\Pi \sqsupseteq (U;D)\} \\ & \Rightarrow & Y \sqsupseteq (X;D) \qquad\qquad \{(D;U) \sqsupseteq \Pi\} \\ & \Rightarrow & (Y;U) \sqsupseteq X \end{array}$$

which together with the Definition 4.3.1 implies that

$$X/U \;=\; (X;D), \qquad \text{for all } X$$

By taking $X \;=\; \Pi$ we obtain $D \;=\; \Pi/U$ which leads to the conclusion

$$U/U \;=\; U;D \;=\; U;(\Pi/U) \qquad \Box$$

Simulations have their most significant application in the concrete representation of abstract data types [87]. An algorithm is often most clearly described as a program operating on variables whose values range over abstract mathematical spaces like sets and functions. But computers do not provide direct methods of storing and operating on such values. Instead, they provide operations on concrete data – bits, bytes, arrays, addresses and files. So before the abstract program can be run, the abstract data must be conceptually translated to the concrete data structure which represents it, and the operations on the abstract data must be transformed to corresponding actions on the concrete data in a completely consistent way. This is done by simulations.

Let the abstract program take the form $F(A_1, A_2, \ldots)$, where the A_i are the collection of all primitive operations which update the abstract data, and F is constructed using the notations of the programming language. In the process of top-down development, we need to find a simulation $(D,\, U)$, where D is a relation between abstract data values and their concrete representation, whereas U maps the data "upwards" from concrete to abstract. Now the abstract program $F(A_1, A_2, \ldots)$ can be translated to the concrete program

$$D; F((U; A_1; D),\ (U; A_2; D), \ldots); U$$

where the initial D first translates abstract data to concrete, the translated program manipulates the concrete data, and the final U translates its concrete results back up to the abstract data afterwards. The individual operations $U; A_i; D$ are then taken as specifications for optimised programs operating directly by available machine operations on the actually stored concrete data.

Example 4.4.6 (Binary representation)

A natural number n is represented as an array b of bits. The components of the simulation are

$$\begin{aligned} U &= n := \Sigma_i(b_i \times 2^i) \\ D &= \mathbf{true} \vdash \bigwedge_i (b'_i = (n \div 2^i) mod\ 2) \end{aligned}$$

An abstract operation might be

$$A = (n := n + 1)$$

The corresponding concrete operation $(U; A; D)$ is

$$\mathbf{true} \vdash \wedge_i\, (b'_i = ((1 - b_i) \lhd \wedge\{k : k < i : b_k = 1\} \rhd b_i)) \qquad \square$$

Example 4.4.7 (Machine addresses)

One of the important tasks of a compiler is to replace the identifiers of the symbolic program by the numeric addresses of locations in the machine store. Suppose that Φ is a symbol table that maps each variable name of the symbolic program to the address of the main memory M allocated to hold its value. So $M[\Phi x]$ is the location holding the value of x. Clearly it is necessary to insist that Φ is total and injective. We define U as a sort of symbolic dump; it assigns to each variable the value in the corresponding location.

$$\begin{aligned} U &=_{df}\ \mathbf{var}\, x, y, \ldots, z\, ; (x, y, \ldots, z := M[\Phi x], M[\Phi y], \ldots, M[\Phi z])\, ; \mathbf{end}\, M \\ D &=_{df}\ \mathbf{var}\, M\, ; (M[\Phi x], M[\Phi y], \ldots, M[\Phi z] := x, y, \ldots, z)\, ; \mathbf{end}\, x, y, \ldots, z \end{aligned}$$

It is not difficult to prove that (D, U) is a simulation using the algebraic laws of Chapter 2. $\square$

Of course, the translation from the abstract to the concrete version of the program must be correct in the sense that it can only improve the result, that is

$$F(A_1, A_2, \ldots) \sqsubseteq (D; F(U; A_1; D,\ U; A_2; D, \ldots); U)$$

Define

$$L \ =_{df} \ pre_U \circ post_D$$
$$R \ =_{df} \ pre_D \circ post_U$$

Because of the cancellation properties of the Galois connection $(L,\ R)$ (Theorem 4.4.3), the correctness criterion may be rewritten

$$R(F(A_1, A_2, \ldots)) \sqsubseteq F(R(A_1),\ R(A_2), \ldots)$$

or more briefly, R is an $F_{\sqsubseteq}$-morphism. Rather than proving this separately for each simulation, we will give a general result that can be applied to all programs.

Since F has been written solely in the notations of the programming language, it is sufficient to prove that R is a $\Sigma_{\sqsubseteq}$-morphism. Recursion is also allowed, by the μ-fusion theorem (Theorem 4.2.12).

Theorem 4.4.8 (Piecewise data refinement)

Let v and w be the program variables used in the abstract and concrete versions of the program respectively. If

$$U \ =_{df} \ \mathbf{var}\, v := f(w);\ \mathbf{end}\, w$$

then U is a co-simulation and the corresponding R is a $\{\sqcap,\ ;\ ,\ \lhd b \rhd\}_{\sqsubseteq}$-morphism.

(1) $R(P \sqcap Q) \ = \ R(P) \sqcap R(Q)$

(2) $R(P \lhd b(v) \rhd Q) \ = \ R(P) \lhd b(f(w)) \rhd R(Q)$

(3) $R(P; Q) \ \sqsubseteq \ R(P); R(Q)$ □

Exercise 4.4.9

Prove that if $(D,\ U)$ is a simulation, then D is a left factor of $I\!I$ and U is a right factor of $I\!I$. □