Chapter 2

Formal Logic

The methods in this textbook for reasoning about programs are based on using formal logic to characterize program execution. Here, we review some rudiments of logic and show how logic can be used to formalize safety and liveness. Our study of logic is done from the programmer's viewpoint, not the logician's. For us, logic is simply a tool. However, as with most tools, it must be understood to be used effectively.

2.1 Formal Logical Systems

A *formal logical system* consists of

- a set of symbols,
- a set of *formulas* constructed from the symbols,
- a set of distinguished formulas, called *axioms*, and
- a set of inference rules.

The set of formulas is called the *language* of the logic; it is defined by a giving a syntax for constructing formulas from the symbols.

The *inference rules* of the logic allow formulas to be derived from other formulas. We specify inference rules using the notation

$$\frac{P_1,\ P_2,\ ...,\ P_n}{C}$$

where *premises* $P_1, P_2, ..., P_n$ and *conclusion* C are either formulas or schematic representations of formulas. An inference rule can be used to derive a formula F from formulas $F_1, ..., F_n$ if F is an instantiation of the conclusion of the rule and $F_1, ..., F_n$ are instantiations of its premises. We give an example shortly.

A *proof* in a formal logical system is a sequence of formulas in which each formula is an axiom or can be derived by using an inference rule whose premises are axioms or previous formulas in the sequence. Any formula in a proof is called a *theorem*. Thus, theorems are special only in that they are axioms or the result of applying inference rules to axioms and other theorems. We use the notation $\vdash_L F$ to assert that F is a theorem of logic L and the notation $A_1, A_2, ..., A_n \vdash_L F$ to assert that F is a theorem of the logical system resulting when A_1 through A_n are added to L as axioms.

To illustrate these ideas, we investigate a simple formal logical system, PQ–L:

Symbols: P, Q, –

Formulas: Formulas have the form $a\,\mathrm{P}\,b\,\mathrm{Q}\,c$, where a, b, and c each represents a finite sequence of zero or more dashes ("–").

Axioms:

(2.1) – P – Q – –

(2.2) – – P – Q – – –

Inference Rule:

(2.3) $$\frac{a\,\mathrm{P}\,b\,\mathrm{Q}\,c,\quad d\,\mathrm{P}\,e\,\mathrm{Q}\,f}{a\,d\,\mathrm{P}\,b\,e\,\mathrm{Q}\,c\,f}$$

Examples of PQ–L formulas are:

– – – P – Q – –
P Q –
– P – Q – –

We now give an example of a PQ–L proof. Strictly speaking, the justification delimited by « and » that precedes each formula is not part of the proof. However, such justifications simplify reading the proof, so they are usually included.

«Axiom (2.1)»
1. – P – Q – –

«Axiom (2.2)»
2. – – P – Q – – –

«Rule (2.3) with 1 and 2»
3. – – – P – – Q – – – – –

«Rule (2.3) with 1 and 3»
4. – – – – P – – – Q – – – – – – –

Formulas 1 and 2 are axioms of PQ–L; formula 3 is the result of using inference rule (2.3) with formulas 1 and 2 as premises; and formula 4 results from using inference rule (2.3) with formulas 1 and 3 as premises. Note that this proof contains four theorems—one theorem on each line.

Models and Interpretations

A digital computer could be instructed in the rules of some formal logic and would thereafter be able to prove theorems of the logic. Such theorems would probably not be very useful, though. A formal logical system stops being an exercise in symbol pushing and becomes a powerful tool when its theorems are true statements of some domain of discourse that concerns us. An *interpretation* for a logic L gives a meaning to formulas of L in terms of a domain of discourse by mapping each formula to *true* or *false*. For example, we might be interested in statements about the domain "integers and addition," and we would expect an interpretation to map formula $1+3=4$ to *true* and $1+4=6$ to *false*.

Interpretations usually deal with some class of mathematical objects, such as sets, relations, mappings, or sequences. For example, when program states are the domain of discourse, the class of possible interpretations is the set of mappings from program variables to values; when program executions are the domain of discourse, the class might be sequences of such mappings.

An interpretation ι is a *model for a formula* P, denoted by $\iota \models P$, if P is mapped to *true* by ι. Thus, an interpretation that maps the symbols "0" to zero, "1" to one, "+" to the function we know as integer addition, and "=" to equality is a model for the formula $1+0=1$, but so is an interpretation that instead maps "0" to *false*, "1" to *true*, and "+" to logical-or. Not only can a formula have more than one model, but a given interpretation can be a model for more than one formula. For example, both of the interpretations just described are models for $0+1=1$, $0+0=0$, $1+0=1$, and many other formulas.

An interpretation ι is a *model for a logic* L if it is a model for every theorem of L. For example, PQ–L has the following model:

(2.4) **Addition-Equality Interpretation.** A formula $a\,\mathrm{P}\,b\,\mathrm{Q}\,c$ is mapped to *true* iff $|a| + |b| = |c|$, where $|x|$ is the number of dashes in sequence x. □

To see that Addition-Equality Interpretation (2.4) is a model for PQ–L, first note that it is a model for each PQ–L axiom. (Under the interpretation, axiom (2.1) asserts $1+1=2$ and (2.2) asserts $2+1=3$.) Next, observe that formulas obtained using inference rule (2.3) have Addition-Equality Interpretation (2.4) as a model, because if $a+b=c$ and $d+e=f$ hold, then so does $(a+d)+(b+e)=(c+f)$. Thus, all theorems of PQ–L have Addition-Equality Interpretation (2.4) as a model.

The following interpretation is also a model for PQ–L.

(2.5) **Addition-Inequality Interpretation.** A formula $a\,\mathrm{P}\,b\,\mathrm{Q}\,c$ is mapped to *true* iff $|a| + |b| \le |c|$, where $|x|$ is the number of dashes in sequence x. □

This interpretation is not the same as Addition-Equality Interpretation (2.4). Some formulas have Addition-Inequality Interpretation (2.5) as a model but not Addition-Equality Interpretation (2.4). An example of such a formula is:

(2.6) – P – – Q – – – –

If every element in some set R of interpretations is a model for P, then P is said to be *R-valid*. This is denoted by $R \models P$. When R is clear from the context, it may be omitted. A formula P is *R-satisfiable* if at least one interpretation in R is a model for P. Observe that R-satisfiable formulas need not be R-valid and that R-valid formulas need not be theorems. For example, consider any set of interpretations containing Addition-Equality Interpretation (2.4) and Addition-Inequality Interpretation (2.5). PQ–L formula (2.6) is then satisfiable but not valid because Addition-Inequality Interpretation (2.5) is a model but Addition-Equality Interpretation (2.4) is not.

An important attribute of a logic is soundness relative to a set of interpretations R. An axiom is *R-sound* iff it is R-valid. An inference rule is *R-sound* iff any formula derived using that rule is R-valid whenever all its premises are R-valid. And a logic is *R-sound* iff all of its axioms and inference rules are. If a logic is R-sound, then facts about a domain of discourse characterized by R can be deduced in a purely mechanical fashion—theorems of the logic are derived mechanically by applying inference rules, and soundness means that theorems are true statements about the domain of discourse.

Soundness of a logic is rarely an accident. A logic is usually intended to facilitate reasoning about a given domain of discourse, so the logic is defined with a particular set R of interpretations in mind. Each axiom is defined so that all interpretations in R are models; each inference rule is formulated so that any formula derived using it will be R-valid whenever its premises are R-valid.

A logic L is *complete* iff every R-valid formula is a theorem. If we let *FACTS* be the set of R-valid formulas of L and *THMS* be the set of theorems, then soundness means that $THMS \subseteq FACTS$ and completeness means that $FACTS \subseteq THMS$. For example, Addition-Inequality Interpretation (2.5) is a model for formula (2.6), but (2.6) is not provable in PQ–L, so PQ–L is not complete with respect to formulas that are valid according to Addition-Inequality Interpretation (2.5). A logic that is both sound and complete allows exactly the R-valid formulas to be proved. Our failure to prove that an R-valid formula is a theorem in such a logic cannot be attributed to weakness of the logic.

Unfortunately, the domains of discourse of concern to us—arithmetic truths, program behavior, and so on—do not have sound and complete axiomatizations. This is a consequence of Gödel's incompleteness theorem, which states that no formal logical system that axiomatizes arithmetic can be both sound and complete.[1] Fortunately, this incompleteness is not a problem in practice. The theorems we will have to prove when reasoning about most programs are ones for which proofs can be constructed with ease.

[1]More precisely, no logical system in which the set of axioms is recursive can provide a sound and complete axiomatization of arithmetic.

In order to isolate sources of incompleteness in a logic, the logic can be defined in a hierarchical fashion. Logic L is an *extension* of logic L' if the symbols, formulas, axioms, and inference rules of L' are included in L. We say that L is *complete relative to* L' if adding as axioms to L every R-valid formula of L' results in a complete formal logical system. If a logic L is complete relative to L', then L introduces no source of incompleteness beyond that already in L'. The logics we will define for reasoning about programs are extensions of logics for reasoning about integers under the arithmetic operations, sequences under the usual sequence operations, and so on. In light of Gödel's incompleteness theorem, the best we can then hope for is relative completeness.

A final important property for a logic is *expressiveness*. A logic is *expressive* with respect to a domain of discourse insofar as it allows statements about that domain of discourse to be expressed as formulas. For example, the syntax of PQ–L does not allow statements like $3+1=1+3$ to be made. PQ–L is not very expressive.

Formal and Informal Proofs

The point of a proof is to provide convincing evidence of the correctness of some statement that is expressed as a formula. What is convincing evidence? Imagine a logical system for which the soundness of each axiom and inference rule can be accepted without question. A formal proof using such a logic will constitute convincing evidence, because each step in the proof is, by supposition, truth-preserving. Moreover, although such a proof might be tedious, it can be checked mechanically.

Consider an informal proof written as English text. Although perhaps easier to read, such a proof might leave steps out or contain subtle errors, since English is rich but ambiguous. Thus, one might argue that such a text cannot be construed as convincing evidence.

On the other hand, a good informal proof can be viewed as an outline, or set of instructions, for constructing a formal proof in some specified formal logical system. In that case, such an informal proof might well be considered convincing evidence. Informal proofs are legitimate reasoning tools when they are constructed to serve as descriptions of formal proofs.

2.2 Propositional Logic

Propositional Logic, the first nontrivial formal logical system we study, is the basis for all the other logical systems discussed in this text. It is a formalization of the type of reasoning most would call common sense. A *proposition* is a statement that is either *true* or *false*. In Propositional Logic, *propositional variables* are used to denote propositions, and *propositional connectives* are used to form formulas from propositional variables and propositional constants.

Various sound and complete axiomatizations of Propositional Logic have been proposed. The one described below has the virtue of brevity; it is impressive how powerful such a small logical system can be.

Symbols: Propositional connective: $\Rightarrow$
Propositional constant: *false*
Propositional variables: p, q, r, ...
Grouping symbols: (,)

Formulas: *false* is a formula.

Each propositional variable is a formula.

For P and Q formulas: $(P \Rightarrow Q)$ is a formula; P is called the *antecedent* and Q the *consequent*. Parentheses may be omitted when no ambiguity results.

Axioms:

(2.7) *Affirmation of the Consequent*: $p \Rightarrow (q \Rightarrow p)$

(2.8) *Self-Distributive Law of Implication*:
$$(r \Rightarrow (p \Rightarrow q)) \Rightarrow ((r \Rightarrow p) \Rightarrow (r \Rightarrow q))$$

(2.9) *Double Negation*: $(((p \Rightarrow \mathit{false}) \Rightarrow \mathit{false}) \Rightarrow p)$

Inference Rules:

(2.10) *Modus Ponens*: $\dfrac{P,\ (P \Rightarrow Q)}{Q}$

(2.11) *Substitution*: Let P^q_Q denote the formula obtained by substituting formula Q for every occurrence of propositional variable q in formula P. Then:
$$\frac{P}{P^q_Q}$$

The following proof in Propositional Logic establishes that $p \Rightarrow p$ is a theorem:

«Self-Distributive Law of Implication (2.8)»
1. $(r \Rightarrow (p \Rightarrow q)) \Rightarrow ((r \Rightarrow p) \Rightarrow (r \Rightarrow q))$

«Substitution (2.11) into 1 using s for p»
2. $(r \Rightarrow (s \Rightarrow q)) \Rightarrow ((r \Rightarrow s) \Rightarrow (r \Rightarrow q))$

«Substitution (2.11) into 2 using p for q»
3. $(r \Rightarrow (s \Rightarrow p)) \Rightarrow ((r \Rightarrow s) \Rightarrow (r \Rightarrow p))$

«Substitution (2.11) into 3 using p for r»
4. $(p \Rightarrow (s \Rightarrow p)) \Rightarrow ((p \Rightarrow s) \Rightarrow (p \Rightarrow p))$

«Substitution (2.11) into 4 using q for s»
5. $(p \Rightarrow (q \Rightarrow p)) \Rightarrow ((p \Rightarrow q) \Rightarrow (p \Rightarrow p))$

«Affirmation of the Consequent (2.7)»
6. $p \Rightarrow (q \Rightarrow p)$

«Modus Ponens (2.10) using 6, 5»
7. $(p \Rightarrow q) \Rightarrow (p \Rightarrow p)$
«Substitution (2.11) into 7 using $q \Rightarrow p$ for q»
8. $(p \Rightarrow (q \Rightarrow p)) \Rightarrow (p \Rightarrow p)$
«Modus Ponens (2.10) using 6, 8»
9. $p \Rightarrow p$

In addition to the propositional constant and connective defined above, another propositional constant, *true*, and other connectives, "$\neg$" (read "not") for logical negation, "$\vee$" (read "or") for disjunction, "$\wedge$" (read "and") for conjunction, and "$=$" (read "equals") to denote equivalence, can be viewed as abbreviations, according to:

true:	$false \Rightarrow false$
$\neg P$:	$P \Rightarrow false$
$P \vee Q$:	$(P \Rightarrow Q) \Rightarrow Q$
$P \wedge Q$:	$\neg (\neg P \vee \neg Q)$
$P = Q$:	$(P \Rightarrow Q) \wedge (Q \Rightarrow P)$

When parentheses are omitted, the connectives are assumed to have precedence given by these binding strengths:

$\neg$	$=$	$\wedge$	$\vee$	$\Rightarrow$
Tightest				Weakest

In addition, all except the equals connective are assumed to be left associative. For example:

$P \wedge Q \wedge R$	is an abbreviation for	$(P \wedge Q) \wedge R$
$\neg P \Rightarrow Q$	is an abbreviation for	$(\neg P) \Rightarrow Q$
$P \vee Q \wedge R$	is an abbreviation for	$P \vee (Q \wedge R)$

The equals connective is assumed to be *conjunctional*, which means that $P = Q = R$ is an abbreviation for $(P = Q) \wedge (Q = R)$.

Formulas of Propositional Logic are interpreted by functions, called *states*, that map each propositional variable to *true* or *false*. The value of a propositional variable p in a state s is denoted by $s[\![p]\!]$.

(2.12) **Interpretation for Propositional Logic.** For a propositional variable p and state s:

$s \models p$ iff $s[\![p]\!]$ equals *true*

For Propositional Logic formulas P and Q and state s:

$s \models (\neg P)$ iff $s \not\models P$

$s \models (P \vee Q)$ iff $s \models P$ or $s \models Q$

$s \models (P \wedge Q)$ iff $s \models P$ and $s \models Q$

$s \models (P \Rightarrow Q)$ iff $s \not\models P$ or $s \models Q$

$s \models (P = Q)$ iff $s \models P$ equals $s \models Q$ □

The value of a formula P in a state s need not be computed recursively using Interpretation for Propositional Logic (2.12); instead, it can be computed as follows. First, each propositional variable p in P is replaced by its value $s[\![p]\!]$ in s. The resulting formula is then simplified by repeatedly selecting a subexpression that involves a single propositional connective and replacing it with a propositional constant, according to the table below. This process is repeated until it can no longer be carried out; it yields a single propositional constant. That propositional constant is taken to be the value of the original formula P.

The following table gives the values for subexpressions that contain a single propositional connective. Each row of the table corresponds to possible values for the constant(s) in the subexpression; each column corresponds to a subexpression containing a single propositional connective. The value of a subexpression is the value that appears in the row and column corresponding to the propositional constant(s) and the subexpression being simplified.

(2.13) **Meaning of Propositional Connectives:**

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \Rightarrow q$	$p = q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

For example, the value of $\neg p \Rightarrow (p \vee q)$ in a state where p is *false* and q is *true* is computed as follows.

$\neg$ *false* $\Rightarrow$ (*false* $\vee$ *true*)	replacing variables by their values.
true $\Rightarrow$ (*false* $\vee$ *true*)	replacing $\neg$ *false*.
true $\Rightarrow$ *true*	replacing *false* $\vee$ *true*.
true	replacing *true* $\Rightarrow$ *true*.

Some Convenient Enhancements

Although the axiomatization given above for Propositional Logic is complete, using it can be awkward. Completeness of an axiomatization is a far cry from convenience. Therefore, we augment the axiomatization to allow simpler proofs of theorems that tend to arise in practice—theorems of the form $P = Q$ and $P \Rightarrow Q$, where P and Q are formulas of Propositional Logic.

The laws[2] below are formulated in terms of propositional variables p, q, and r. Substitution (2.11) can be used to replace these variables by arbitrary propositional formulas.

(2.14) *Commutative Laws*: (a) $(p \wedge q) = (q \wedge p)$
(b) $(p \vee q) = (q \vee p)$
(c) $(p = q) = (q = p)$

(2.15) *Associative Laws*: (a) $(p \wedge (q \wedge r)) = ((p \wedge q) \wedge r)$
(b) $(p \vee (q \vee r)) = ((p \vee q) \vee r)$

(2.16) *Distributive Laws*: (a) $(p \wedge (q \wedge r)) = ((p \wedge q) \wedge (p \wedge r))$
(b) $(p \wedge (q \vee r)) = ((p \wedge q) \vee (p \wedge r))$
(c) $(p \vee (q \wedge r)) = ((p \vee q) \wedge (p \vee r))$
(d) $(p \vee (q \vee r)) = ((p \vee q) \vee (p \vee r))$
(e) $(p \vee (q = r)) = ((p \vee q) = (p \vee r))$
(f) $(p \vee (q \Rightarrow r)) = ((p \vee q) \Rightarrow (p \vee r))$
(g) $(p \Rightarrow (q \wedge r)) = ((p \Rightarrow q) \wedge (p \Rightarrow r))$
(h) $(p \Rightarrow (q \vee r)) = ((p \Rightarrow q) \vee (p \Rightarrow r))$
(i) $(p \Rightarrow (q = r)) = ((p \Rightarrow q) = (p \Rightarrow r))$
(j) $(p \Rightarrow (q \Rightarrow r)) = ((p \Rightarrow q) \Rightarrow (p \Rightarrow r))$

(2.17) *De Morgan's Laws*: (a) $\neg (p \wedge q) = (\neg p \vee \neg q)$
(b) $\neg (p \vee q) = (\neg p \wedge \neg q)$

(2.18) *Identity Law*: $p = p$

(2.19) *Negation Law*: $p = \neg \neg p$

(2.20) *Excluded Middle Law*: $(p \vee \neg p) = \mathit{true}$

(2.21) *Contradiction Law*: $(p \wedge \neg p) = \mathit{false}$

---

[2]Here and throughout, we label as "laws" useful theorems that are not axioms.

(2.22) *Implication Laws*: (a) $(p \Rightarrow q) = (\neg p \vee q)$
(b) $(p \Rightarrow q) = ((p \wedge q) = p)$

(2.23) *Equality Law*: $(p = q) = ((p \Rightarrow q) \wedge (q \Rightarrow p))$

(2.24) *Equality-Simplification Law*: $p = (p = true)$

(2.25) *Or-Simplification Laws*: (a) $(p \vee p) = p$
(b) $(p \vee true) = true$
(c) $(p \vee false) = p$
(d) $(p \vee (p \wedge q)) = p$

(2.26) *And-Simplification Laws*: (a) $(p \wedge p) = p$
(b) $(p \wedge true) = p$
(c) $(p \wedge false) = false$
(d) $(p \wedge (p \vee q)) = p$

(2.27) *Implication-Simplification Laws*: (a) $(p \Rightarrow true) = true$
(b) $(true \Rightarrow p) = p$

(2.28) *Importation/Exportation Law*: $(p \Rightarrow (q \Rightarrow r)) = ((p \wedge q) \Rightarrow r)$

(2.29) *Antecedent-Strengthening Law*: $(p \wedge q) \Rightarrow p$

(2.30) *Consequent-Weakening Law*: $p \Rightarrow (p \vee q)$

(2.31) *Implication-Deduction Laws*: (a) $p \wedge (p \Rightarrow q) \Rightarrow q$
(b) $\neg q \wedge (p \Rightarrow q) \Rightarrow \neg p$

(2.32) *Contrapositive Law*: $(p \Rightarrow q) = (\neg q \Rightarrow \neg p)$

(2.33) *Constructive Dilemma Laws*:
(a) $(p \Rightarrow q) \wedge (r \Rightarrow s) \Rightarrow ((p \wedge r) \Rightarrow (q \wedge s))$
(b) $(p \Rightarrow q) \wedge (r \Rightarrow s) \Rightarrow ((p \vee r) \Rightarrow (q \vee s))$

Some additional inference rules will also be convenient. These rules do not allow new theorems to be proved, but they do simplify the construction and presentation of proofs.

The first rules assert that equals can be substituted for equals. As we shall see, this supports an equational style of reasoning that is similar to the familiar one used for manipulating algebraic formulas.

(2.34) *Substitution of Equals*: For any propositional variable p:

$$\text{(a)}\ \frac{Q = R}{P^p_Q = P^p_R} \qquad \text{(b)}\ \frac{Q = R}{P^p_R = P^p_Q}$$

Generalizing from Substitution of Equals (2.34), one might try to infer $P^p_Q \Rightarrow P^p_R$ from a theorem $Q \Rightarrow R$. However, such an inference could be unsound. As an example, consider the case where P is $p \Rightarrow q$ and premise $Q \Rightarrow R$ is the theorem $a \Rightarrow (a \vee b)$. $P^p_Q \Rightarrow P^p_R$ would be $(p \Rightarrow q)^p_a \Rightarrow (p \Rightarrow q)^p_{a \vee b}$, which is $(a \Rightarrow q) \Rightarrow ((a \vee b) \Rightarrow q)$ and is not valid. But notice that it is sound to conclude $(p \Rightarrow q)^q_a \Rightarrow (p \Rightarrow q)^q_{a \vee b}$ (i.e., $(p \Rightarrow a) \Rightarrow (p \Rightarrow (a \vee b))$) from theorem $a \Rightarrow (a \vee b)$.

We seek a sound inference rule that, given a theorem $Q \Rightarrow R$, yields a formula involving two instances of P—one instance with some propositional variable p replaced by Q and the other with p replaced by R. Towards this end, we define the *parity* of a propositional variable in a formula.

(2.35) **Parity of a Variable.** For P a Propositional Logic formula and p a propositional variable not appearing in an operand of an equivalence[3] in P:

- p has *even* parity in P iff each occurrence of p is within an even number of negations and antecedents of implications.
- p has *odd* parity in P iff each occurrence of p is within an odd number of negations and antecedents of implications. □

For example, in $p \Rightarrow q$, p has odd parity and q has even parity; in $\neg (p \Rightarrow q)$, p has even parity and q has odd parity.

If $Q \Rightarrow R$ is a theorem, then $P^p_Q \Rightarrow P^p_R$ is valid if p has even parity in P, and $P^p_R \Rightarrow P^p_Q$ is valid if p has odd parity in P. This can be shown by structural induction on P and is the basis for the following inference rules.

(2.36) *Monotonicity Rule*: For a propositional variable p that has even parity in Propositional Logic formula P:

$$\frac{Q \Rightarrow R}{P^p_Q \Rightarrow P^p_R}$$

(2.37) *Antimonotonicity Rule*: For a propositional variable p that has odd parity in Propositional Logic formula P:

$$\frac{Q \Rightarrow R}{P^p_R \Rightarrow P^p_Q}$$

[3]Recall, any equivalence $A = B$ is equal to $(A \Rightarrow B) \wedge (B \Rightarrow A)$, so precluding equivalences does not really constitute a restriction.

Monotonicity Rule (2.36) allows us to conclude from premise $a \Rightarrow (a \vee b)$ that $(p \Rightarrow a) \Rightarrow (p \Rightarrow (a \vee b))$ and $(\neg a \Rightarrow q) \Rightarrow (\neg (a \vee b) \Rightarrow q)$ are theorems, and Antimonotonicity Rule (2.37) allows us to conclude that $((a \vee b) \Rightarrow q) \Rightarrow (a \Rightarrow q)$ is a theorem.

The next inference rules assert that equality and implication are transitive. They are useful for combining formulas whose principal connective is equals and/or implies.

(2.38) *Transitivity of Equality*: $\dfrac{P = Q,\ Q = R}{P = R}$

(2.39) *Transitivity of Implication*:

(a) $\dfrac{P \Rightarrow Q,\ Q \Rightarrow R}{P \Rightarrow R}$ (b) $\dfrac{P = Q,\ Q \Rightarrow R}{P \Rightarrow R}$ (c) $\dfrac{P \Rightarrow Q,\ Q = R}{P \Rightarrow R}$

It now becomes possible to write proofs and proof steps in an equational format. This *equational format* consists of a sequence of formulas, each on a separate line, with adjacent lines being separated by equals or implies, along with a justification (delimited by « and »). An example is the following:

(2.40)
$$\begin{array}{ll} & A \\ = & \quad \text{«Why } A = B \text{ is a theorem»} \\ & B \\ \Rightarrow & \quad \text{«Why } B \Rightarrow C \text{ is a theorem»} \\ & C \\ = & \quad \text{«Why } C = D \text{ is a theorem»} \\ & D \end{array}$$

In our equational proofs, a justification "Why X is a theorem" gives only the premise that enables X to be proved whenever the inference rule being used to make that deduction is obvious—and it usually is obvious.

(2.41) **Equational Proof Justifications.** A justification "Why X is a theorem" in an equational proof must be:

(i) The theorem that enables X to be deduced by repeated uses of Substitution (2.11). Details of the substitutions may be omitted when they are obvious.

(ii) The theorem that enables X to be deduced according to one of the Substitution of Equals (2.34) rules.

(iii) The theorem that enables X to be deduced according to Monotonicity Rule (2.36).

(iv) The theorem that enables X to be deduced according to Antimonotonicity Rule (2.37).

Without mention, conjuncts or disjuncts in formulas may be reordered, thereby omitting steps that involve Commutative Laws (2.14). □

Equational proof (2.40) establishes $A \Rightarrow D$. Formally, that conclusion follows from repeated use of Transitivity of Equality (2.38) and Transitivity of Implication (2.39) on justifications $A = B$, $B \Rightarrow C$, and $C = D$. But that is not all that can be deduced from (2.40).

(2.42) **Equational Proof Conclusions.**

(i) Given a proof of $A \Rightarrow D$ with A a theorem, D is also a theorem.

(ii) Given a proof of $A = D$ with A a theorem, D is also a theorem.

(iii) Given a proof of $A = D$ with D a theorem, A is also a theorem. □

The formal justification for part (i) of Equational Proof Conclusions (2.42) is a single Modus Ponens (2.40) step. To justify part (ii), three steps suffice—an equational proof and two steps involving Modus Ponens (2.40).

1. $A = D$
$=$ «Equality Law (2.23)»
$(A \Rightarrow D) \wedge (D \Rightarrow A)$
$\Rightarrow$ «Antecedent-Strengthening Law (2.29)»
$(A \Rightarrow D)$

«Modus Ponens (2.40) with $A = D$ (assumed previously proved) and 1»

2. $A \Rightarrow D$

«Modus Ponens (2.40) with A (assumed previously proved) and 2»

3. D

The justification of part (iii) is similar to that just given for part (ii).

In order to illustrate the virtues of the equational proof format, consider proving:

(2.43) $(InA \Rightarrow \neg\, InB) \;=\; \neg\, (InA \wedge InB)$

If InA and InB denote propositions

InA: "process A is executing in its critical section"

InB: "process B is executing in its critical section"

then a proof of (2.43) would establish that processes A and B are not both executing at the same time in their critical sections provided that whenever A executes in its critical section, B does not. Here is an equational proof:

$InA \Rightarrow \neg\, InB$
$=$ «Implication Law (2.22a)»

$$\begin{array}{ll} & \neg\, InA \vee \neg\, InB \\ = & \quad \text{«De Morgan's Law (2.17a)»} \\ & \neg\, (InA \wedge InB) \end{array}$$

For comparison, here is a nonequational proof of (2.43).

«Implication Law (2.22a)»
1. $(p \Rightarrow q) \;=\; (\neg\, p \vee q)$

«Substitution (2.11) into 1 using InA for p and $\neg\, InB$ for q»
2. $(InA \Rightarrow \neg\, InB) \;=\; (\neg\, InA \vee \neg\, InB)$

«De Morgan's Law (2.17a)»
3. $\neg\, (p \wedge q) \;=\; (\neg\, p \vee \neg\, q)$

«Substitution of Equals (2.34b) using p for P and 3 for $Q = R$»
4. $(\neg\, p \vee \neg\, q) \;=\; \neg\, (p \wedge q)$

«Substitution (2.11) into 4 using InA for p and InB for q»
5. $(\neg\, InA \vee \neg\, InB) \;=\; \neg\, (InA \wedge InB)$

«Transitivity of Equality (2.38) with 2, 5»
6. $(InA \Rightarrow \neg\, InB) \;=\; \neg\, (InA \wedge InB)$

Decision Procedure for Propositional Logic

In any sound and complete axiomatization of a logic, a formula is valid iff it is a theorem. This fact can be used to determine whether a formula of Propositional Logic is a theorem: simply determine whether the formula is valid.[4]

(2.44) **Decision Procedure for Propositional Logic.** If the value of formula P is *true* in every possible state, then P is a theorem. □

In order to determine validity of a propositional formula that involves N distinct variables, 2^N states must be checked. For example, we show that

$$\neg\, (p \wedge q) = (\neg\, p \vee \neg\, q)$$

is valid by checking 2^2 cases:

p	q	$\neg\, (p \wedge q)$	$\neg\, p \vee \neg\, q$	$\neg\, (p \wedge q) = (\neg\, p \vee \neg\, q)$
false	*false*	*true*	*true*	*true*
false	*true*	*true*	*true*	*true*
true	*false*	*true*	*true*	*true*
true	*true*	*false*	*false*	*true*

[4]The valid formulas in Propositional Logic are sometimes called *tautologies*.

2.3 A Predicate Logic

We now turn attention to *Predicate Logic*, a logic that is considerably more expressive than Propositional Logic. Predicate Logic augments Propositional Logic in three ways:

- A new class of variables is introduced to denote values other than *true* and *false*.
- Predicates and functions allow properties of values to be expressed.
- Quantification allows properties about sets of values to be expressed.

This makes the logic well suited for characterizing relationships among program variables.

Various formulations of Predicate Logic have been proposed. The one given below is an extension of Propositional Logic. Thus, it contains all the formulas, axioms, and inference rules given in §2.2. Although not as terse as some axiomatizations of Predicate Logic, it is convenient for reasoning about program states.

Formulas

In our Predicate Logic, *individual variables*, henceforth simply called variables (in contrast to propositional variables), denote values from a fixed domain that includes booleans (i.e., the constants *true* and *false*), integers, reals, sequences, and other values commonly found in programming. *Terms* are defined to be constants, variables, derived terms (discussed below), and function applications (often denoted by infix operators) that involve zero or more terms. The set of functions and operators is presumed to include all those permitted in expressions of the programming language at hand. The value of a term T in a state s, denoted by $s[\![T]\!]$, is the value that results from replacing all variables in T by their values in s and applying the designated function(s).

Relationships among terms are characterized in Predicate Logic by using predicates. A *predicate* is a function application that always yields a boolean and is specified by giving a *predicate symbol* and a parenthesized[5] list of zero or more terms: for p a predicate symbol and T_1, ..., T_n terms, $p(T_1, T_2, ..., T_n)$ denotes a predicate. To improve readability, predicates are sometimes written using an infix notation, as in $x < y$, where "<" is the predicate symbol. Another predicate—different, because it involves different terms—is $x+1 < y+1$. The value of a predicate $p(T_1, T_2, ..., T_n)$ in a state s is either *true* or *false*, based on the meaning of p after $T_1, T_2, ..., T_n$ is each replaced by its value in s.

Predicate symbols in our Predicate Logic include equality[6] (=) along with

[5]The parentheses may be omitted when the list contains zero terms.

[6]In fact, "=" in Propositional Logic was defined to bind tightest of all the binary propositional connectives just so it would have the same precedence as a predicate symbol, and therefore, a single symbol could be employed for both.

all the other standard ones from logic, mathematics, and programming.[7] Rather than fix this set here, we leave it unspecified and introduce predicate symbols as needed. We rely on the reader's familiarity with the usual meaning associated with a predicate symbol instead of giving axioms for predicates constructed using it.

Predicates and terms are defined for all values of their arguments.[8] Predicates always evaluate to *true* or *false*. Terms, on the other hand, are guaranteed to produce a value but are not guaranteed to produce a value in any given set. Thus, we assume that the term x/y, which denotes the result of dividing x by y, has a value even when $y = 0$ or when x and/or y denote non-numeric values. For example, when y is 0, x/y might equal "abc."

We avoid problems associated with having predicates and terms be defined for all values of their arguments by postulating the existence of other predicates to characterize when a predicate or term is meaningful. For this purpose, it is often convenient to be able to assert that the value of a given variable or term is an element of some set. Figure 2.1 gives a useful collection of such sets. In addition, for each term or predicate E, we postulate a predicate def_E that is *true* in exactly those states where E is meaningful. As an example, $def_{x/y}$ might be equivalent to

$$x \in \text{Int} \quad \wedge \quad y \in \text{Int} \quad \wedge \quad y \neq 0$$

Bool	booleans: *false*, *true*
Nat	natural numbers: 0, 1, 2, 3, 4, ...
Int	integers: ..., −3, −2, −1, 0, 1, 2, ...
Rat	rationals: any value x/y, where $x \in \text{Int}$, $y \in \text{Int}$, and $y \neq 0$
Real	reals
Char	characters
V^*	the set of finite-length sequences of values from set V
V^+	the set of nonempty finite-length sequences of values from set V
V^∞	the set of finite-length sequences and infinite-length sequences of values from set V

Figure 2.1. Sets of values

---

[7]Due to Gödel's incompleteness theorem, this means that our Predicate Logic is incomplete, since it axiomatizes arithmetic. The logic, however, is complete relative to arithmetic.

[8]This approach is not satisfactory for reasoning about whether terms and predicates are undefined. But we have no need for such reasoning, so we will not be handicapped. Logics that allow reasoning about whether terms and predicates are undefined usually involve three truth-values—*true*, *false*, and *undefined*—and/or have more complicated axiomatizations than the one given in this chapter.

which is *false* unless x and y are integers and x/y is defined. This means that

(2.45) $def_{x/y} \;\wedge\; z \in \mathrm{Rat} \;\wedge\; (x/y) = z$

is *false* unless x and y are integers, x/y is defined, z is a rational, and $x/y = z$. Therefore, (2.45) is *false* if y is zero, no matter what the (unspecified) value x/y is.

Quantification

It is often useful to be able to assert that values in a given set satisfy some property of interest. We might wish to specify that all values in array $a[1\,..n\,]$ are 0. Conjunction would seem suitable for this purpose:

(2.46) $a[1] = 0 \;\wedge\; a[2] = 0 \;\wedge\; \cdots \;\wedge\; a[n\,] = 0$

Disjunction could be used to assert that some value in a set satisfies a property of interest. Thus,

(2.47) $b[1] = 16 \;\vee\; b[2] = 16 \;\vee\; \cdots \;\vee\; b[m\,] = 16$

would assert that some element in array $b[1\,..m\,]$ equals 16. Unfortunately, this approach is flawed because the ellipses in (2.46) and (2.47) are ambiguous. Formula (2.46) might assert that $a[p\,] = 0$ for all even values of p between 1 and n, or for all values of p that are powers of 2 and at most n, or for all values of p that are prime numbers and at most n, or any number of other things.

Predicate Logic provides *quantified expressions* for unambiguously specifying properties of sets of values. To give a meaning to quantified expressions, it is helpful to have a notation for redefining the value of a variable in a state. For a state s, variable v, and term e, define *augmented state* $(s; v{:}e)$ to be the state that is identical to s, except that the value of v equals the value of e in s:

(2.48) $(s; v{:}e)[\![x]\!]: \begin{cases} s[\![e]\!] & \text{for } \text{“}v\text{”} = \text{“}x\text{”} \\ s[\![x]\!] & \text{for } \text{“}v\text{”} \neq \text{“}x\text{”} \end{cases}$

For example, $(s; w{:}22)[\![w+3]\!]$ equals 25, and $(s; w{:}22)[\![v+3]\!]$ equals $s[\![v+3]\!]$.

Since $(s; v{:}e)$ is itself a state, nested augmented states can be constructed. Thus, state $((s; v{:}9);\ w{:}88)$ associates 88 with variable w, associates 9 with variable v, and otherwise associates values according to s. Composition of augmented states is defined to be left associative, so:

(2.49) $(s; v{:}e; v'{:}e') \;=\; ((s; v{:}e); v'{:}e')$

For example, $(s; v{:}2; v{:}3)[\![v]\!]$ equals 3 and $(s; v{:}2; w{:}v+2)[\![w]\!]$ equals 4.

Predicate Logic has two types of quantified expressions: one for conjunction and one for disjunction. For Predicate Logic formulas R and P and variable

x, the *universally quantified expression* with *range* R and *body* P

(2.50) $(\forall x:\ R:\ P)$

is *true* in a state s iff for *every* value V, $R \Rightarrow P$ is satisfied in state $(s; x{:}\mathrm{V})$. For this reason, (2.50) is read "For all x satisfying R, P holds." An example of a universally quantified expression is

$$(\forall i:\ 1 \le i \le n \ \wedge\ i \in \mathrm{Int}:\ a[i] = 0)$$

which is *true* in a state s iff $a[1], a[2], ..., a[n]$ each equals 0 in state s, just as was intended by (2.46).

An existentially quantified expression specifies that some value in a set satisfies a property of interest. For Predicate Logic formulas R and P, the value of *existentially quantified expression* with *range* R and *body* P

(2.51) $(\exists x:\ R:\ P)$

in a state s is *true* iff for *some* value V, $R \wedge P$ is satisfied in state $(s; x{:}\mathrm{V})$. Not surprisingly, (2.51) is read "There exists an x satisfying R for which P holds." For example, $(\exists i:\ 1 \le i \le m \wedge i \in \mathrm{Int}:\ b[i] = 16)$ is *true* in a state s iff at least one of $b[1], b[2], ..., b[m]$ equals 16 in state s (as was intended by (2.47)).

In a quantified expression, the variable that follows *quantifier* $\forall$ or $\exists$ is called a *bound variable*, and it is associated with that quantifier. The *scope* of a bound variable is defined by the parentheses that delimit the quantified expression in which it is introduced. This is similar to the way scope of identifiers is defined in a block-structured programming language.

An occurrence of a variable v in a formula of Predicate Logic is considered *free* if v is different from every bound variable whose scope contains that occurrence. An occurrence is considered *bound* if v is the same as some bound variable whose scope contains that occurrence. Here are some examples:

(2.52) $i < n$

(2.53) $(\forall i:\ 1 \le i < 10:\ i < n)$

(2.54) $(\forall i:\ 1 \le i < 10:\ i < n) \ \wedge\ i < n$

In (2.52), the occurrences of i and n are free; in (2.53), the three occurrences of i are bound, and the occurrence of n is free; in (2.54), all but the last occurrence of i are bound, the last occurrence is free, and both occurrences of n are free.

Finally, where convenient, we shorten quantified expressions by employing some abbreviations. (Q represents a quantifier "$\forall$" or "$\exists$.")

$(Q\, i:\ P)$ denotes $(Q\, i:\ true:\ P)$

$(Q\, i \in V:\ P)$ denotes $(Q\, i:\ i \in V:\ P)$

$$(Q\, i{\in}V:\ R:\ P) \text{ denotes } (Q\, i:\ i{\in}V \wedge R:\ P)$$

Derived Terms

A facility to associate names with terms gives us the power to extend our Predicate Logic with abstractions tailored for a task at hand. To specify such a *derived term*, we give its name, the set of states in which it is defined, and a method for computing its (unique) value in those states.[9] The syntax we employ for defining a derived term Z is:

$$(2.55)\quad Z:\ \begin{cases} e_1 & \text{if } B_1 \\ \quad \cdots & \\ e_n & \text{if } B_n \end{cases}$$

Each expression and its corresponding guard define a *clause*.

The value of Z in a state s is the value of the unique *expression* e_i for which a corresponding *guard* B_i holds. If no guard holds or more than one guard holds in s, then the value of Z is unspecified.

Notice that the notation we have been using for definitions,

$$Z:\ e$$

can be regarded as an abbreviation for the following definition of a derived term:

$$Z:\ \{\ e \ \text{ if } \mathit{true}$$

As an illustration of a derived term, here is one named M, which equals the maximum of a and b.

$$(2.56)\quad M:\ \begin{cases} a & \text{if } b < a \\ b & \text{if } a \le b \end{cases}$$

Notice that appearances of M in a formula give no hint to the reader that its value depends on a and b. This ability to hide details is important when defining abstractions. But it does create free occurrences of variables in formulas even though those variables do not explicitly appear in the formula. For example, any formula that mentions M may contain free occurrences of a and b.[10] A derived

[9] By convention, derived terms will be named by identifiers starting with an uppercase letter.

[10] The occurrences of a and b are not necessarily free. For example, $(\forall x:\ M \le x)$ contains free occurrences of both a and of b, but $(\forall a:\ M \le a)$ contains bound occurrences of a and free occurrences of b.

term Z contains a *free occurrence* of a variable v whenever an expression or guard containing a free occurrence of v appears in the definition of Z.

By imposing restrictions on the definition of a derived term Z, we ensure that the value of Z is specified when it ought to be. For a term, derived term, or predicate E, let FDT(E) ("free derived terms") be the set of derived terms on which E depends. FDT(E) is defined formally in terms of $fdt(E, ES)$, the union of a set ES of derived terms and the set of derived terms that are free in term E:

FDT(E): $fdt(E, \varnothing)$

The definition of $fdt(E, ES)$ is by induction on the structure of E. It is given in Figure 2.2.

The restrictions on definitions of derived terms are:

(2.57) **Derived Term Restrictions.** A derived term Z defined by (2.55) is *well-defined* provided:

(i) $Z \notin \bigcup_{1 \le i \le n} (\text{FDT}(e_i) \cup \text{FDT}(B_i))$

(ii) $B_1, ..., B_n$ are pairwise-disjoint predicates.

(iii) $(B_1 \Rightarrow def_{e_1}) \wedge \cdots \wedge (B_n \Rightarrow def_{e_n})$ is valid. □

Restriction (i) guarantees termination of the procedure given above for computing the value of Z. The alternative, allowing the value of Z to depend on Z, could lead to infinite recursion. If restriction (ii) is not satisfied, then a state could satisfy both B_i and B_j (for $i \neq j$), and the value of Z might not be uniquely

E	$fdt(E, ES)$
a constant or variable	ES
a derived term, where $E \in ES$	ES
a derived term, where $E \notin ES$ and defined by: E: $\begin{cases} e_1 & \text{if } B_1 \\ \cdots & \\ e_n & \text{if } B_n \end{cases}$	$\bigcup_{1 \le i \le n} (fdt(e_i, ES \cup \{E\}) \cup fdt(B_i, ES \cup \{E\}))$
a function application or predicate $E(T_1, ..., T_n)$	$\bigcup_{1 \le i \le n} fdt(T_i, ES)$

Figure 2.2. *fdt* definition

determined in that state. Restriction (iii) ensures that e_i is defined whenever e_i is used to compute the value of Z.

In light of Derived Term Restrictions (2.57), the value of derived term Z (2.55) is uniquely determined in states that satisfy:

$$def_Z: \quad \bigvee_{1 \le i \le n} B_i$$

Syntax and Interpretation for Predicate Logic

With these preliminaries out of the way, we can now give a syntax and semantics for Predicate Logic. The language of Predicate Logic consists of Propositional Logic formulas and Propositional Logic formulas in which all propositional variables have been replaced by predicates and quantified expressions.

Predicate Logic formulas are interpreted with respect to states. The value of a Predicate Logic formula P in state s is the value of the propositional formula that results from replacing every propositional variable, predicate, and quantified expression in P by its value in s. This is formalized by:

(2.58) **Interpretation for Predicate Logic.** For a predicate $p(T_1, ..., T_n)$ where p is a predicate symbol, $T_1, ..., T_n$ are terms, and s is a state:

$$s \models p(T_1, ..., T_n) \quad \text{iff} \quad p(s[\![T_1]\!], ..., s[\![T_n]\!]) \text{ is } \mathit{true}$$

For Predicate Logic formulas P and Q and state s:

$$s \models (\neg P) \quad \text{iff} \quad s \not\models P$$

$$s \models (P \vee Q) \quad \text{iff} \quad s \models P \quad \text{or} \quad s \models Q$$

$$s \models (P \wedge Q) \quad \text{iff} \quad s \models P \quad \text{and} \quad s \models Q$$

$$s \models (P \Rightarrow Q) \quad \text{iff} \quad s \not\models P \quad \text{or} \quad s \models Q$$

$$s \models (P = Q) \quad \text{iff} \quad s \models P \quad \text{equals} \quad s \models Q$$

$$s \models (\forall x: P: Q) \quad \text{iff} \quad \text{For all V: } (s; x{:}\mathrm{V}) \models (P \Rightarrow Q)$$

$$s \models (\exists x: P: Q) \quad \text{iff} \quad \text{Exists V: } (s; x{:}\mathrm{V}) \models (P \wedge Q) \qquad \square$$

Note (from the final two cases of Interpretation for Predicate Logic (2.58)) that free occurrences of variables obtain their values directly from the state but bound occurrences do not.

Textual Substitution in Predicate Logic

Propositional Logic uses the notation P^x_e to denote the formula that results from substituting e for each occurrence of variable x in P. Thus, the value of P^x_e

in a state s is the same as the value of P in state $(s; x{:}e)$. By taking this to be the defining characteristic of textual substitution, we get:

(2.59) **Semantics for Textual Substitution.** For x a variable, e and T terms, and P a Predicate Logic formula:

(a) $s[\![T^x_e]\!] = (s; x{:}e)[\![T]\!]$

(b) $s \models P^x_e$ iff $(s; x{:}e) \models P$ □

Textual substitution of e for x when T is a constant, variable, or function application is simply a matter of replacing appearances of x with e:

(2.60) *Textual Substitution* [*Constant*]: For a constant C: $C^x_e = C$

(2.61) *Textual Substitution* [*Variable*]: For a variable v:

$$v^x_e = \begin{cases} v & \text{for } \text{“}x\text{”} \neq \text{“}v\text{”} \\ e & \text{for } \text{“}x\text{”} = \text{“}v\text{”} \end{cases}$$

(2.62) *Textual Substitution* [*Term*]: For a function f and terms $T_1, T_2, ..., T_n$:

$$f(T_1, T_2, ..., T_n)^x_e = f((T_1)^x_e, (T_2)^x_e, ..., (T_n)^x_e)$$

It is not difficult to demonstrate that each of these axioms satisfies Semantics for Textual Substitution (2.59) and is, therefore, sound. As an illustration, here is the soundness proof for part of Textual Substitution [Variable] (2.61).

$$\begin{array}{ll} & s[\![v^v_e]\!] \\ = & \quad \text{«Textual Substitution [Variable] (2.61)»} \\ & s[\![e]\!] \\ = & \quad \text{«(2.48)»} \\ & (s; v{:}e)[\![v]\!] \end{array}$$

For a derived term Z, textual substitution of e for x produces another derived term Z^x_e whose definition is like that of Z but with all its clauses modified to reflect the textual substitution:

(2.63) *Textual Substitution* [*Derived Term*]:

$$Z^x_e\text{:} \begin{cases} (e_1)^x_e & \text{if } (B_1)^x_e \\ \quad \cdots & \\ (e_n)^x_e & \text{if } (B_n)^x_e \end{cases}$$

For example, the definition for M^a_b, where M is defined by (2.56), is:

$$M_b^a: \begin{cases} b & \text{if } b < b \\ b & \text{if } b \le b \end{cases}$$

Since $b < b$ is satisfied by no state and $b \le b$ is satisfied by all states, M_b^a is equivalent to b.

Observe that if Z has no free occurrences of x, then for all i, $(e_i)_e^x = e_i$ and $(B_i)_e^x = B_i$. We conclude:

(2.64) *Derived Term Simplification*: For Z with no free occurrences of x:

$$Z_e^x = Z.$$

Thus, according to Derived Term Simplification (2.64), for M defined by (2.56), M_e^v equals M because there are no free occurrences of v in M.

We now turn to axioms concerning textual substitution into formulas of Predicate Logic. For predicates and for formulas constructed using propositional connectives, textual substitution behaves as might be expected.

(2.65) *Textual Substitution* [*Predicate*]: For a predicate symbol p and terms T_1, ..., T_n:

$$p(T_1, ..., T_n)_e^x = p((T_1)_e^x, ..., (T_n)_e^x)$$

(2.66) *Textual Substitution* [*Propositional Connectives*]: For Predicate Logic formulas P and Q:

(a) $(\neg P)_e^x = \neg (P_e^x)$
(b) $(P \wedge Q)_e^x = (P_e^x \wedge Q_e^x)$
(c) $(P \vee Q)_e^x = (P_e^x \vee Q_e^x)$
(d) $(P \Rightarrow Q)_e^x = (P_e^x \Rightarrow Q_e^x)$
(e) $(P = Q)_e^x = (P_e^x = Q_e^x)$

Textual substitution into quantified expressions is tricky because of interactions with bound variables. The naive definition for $(\forall i: R: P)_e^x$ is to perform the designated textual substitutions in R and P, obtaining $(\forall i: R_e^x: P_e^x)$. However, without restricting i, x, and e, it is possible for $(\forall i: R: P)_e^x$ and $(\forall i: R_e^x: P_e^x)$ to have different meanings.

To understand the problem, consider the meanings of:

(2.67) $s \models (\forall i: R: P)_e^x$

(2.68) $s \models (\forall i: R_e^x: P_e^x)$

First, here is the meaning of (2.67).

$s \models (\forall i: R: P)_e^x$
= «Semantics for Textual Substitution (2.59a)»

$(s; x{:}e) \models (\forall i{:}\, R{:}\, P)$
iff «Interpretation for Predicate Logic (2.58)»
For all V: $(s; x{:}e; i{:}\mathrm{V}) \models (R \Rightarrow P)$

Here is the meaning of (2.68).

$s \models (\forall i{:}\, R^x_e{:}\, P^x_e)$
iff «Interpretation for Predicate Logic (2.58)»
For all V: $(s; i{:}\mathrm{V}) \models (R^x_e \Rightarrow P^x_e)$
iff «Textual Substitution [Propositional Connectives] (2.66d)»
For all V: $(s; i{:}\mathrm{V}) \models (R \Rightarrow P)^x_e$
iff «Semantics for Textual Substitution (2.59b)»
For all V: $(s; i{:}\mathrm{V}; x{:}e) \models (R \Rightarrow P)$

Thus, (2.67) and (2.68) are equivalent in a state s iff the value that augmented state $(s; x{:}e; i{:}\mathrm{V})$ assigns to each variable's occurrence in $R \Rightarrow P$ is the same as the value that $(s;\ i{:}\mathrm{V}; x{:}e)$ assigns to that occurrence. In light of (2.48), variables x and i are the only ones whose values might differ; the values assigned to them are summarized in the following table.

x and i distinct?	variable	value of variable in state $(s; x{:}e; i{:}\mathrm{V})$	value of variable in state $(s;\ i{:}\mathrm{V};\ x{:}e)$
no	i	V	$(s; i{:}\mathrm{V})[\![e]\!]$
no	x	V	$(s; i{:}\mathrm{V})[\![e]\!]$
yes	i	V	V
yes	x	$s[\![e]\!]$	$(s; i{:}\mathrm{V})[\![e]\!]$

From the first pair of rows, we see that if x and i are the same variable, then for (2.67) and (2.68) to be equivalent, V and $(s; i{:}\mathrm{V})[\![e]\!]$ must be the same value. Since V is a constant and e is not, there exist states s in which they are not the same. Therefore, one condition for ensuring the equivalence of (2.67) and (2.68) is to rule out textual substitutions where x, the variable being replaced, is not distinct from i, a bound variable in the original formula.

From the second pair of rows, we see that even when x and i are distinct, the value assigned to x by one augmented state is not necessarily the same as the value assigned by the other. In particular, the value assigned to x is either $s[\![e]\!]$ or $(s; i{:}\mathrm{V})[\![e]\!]$. These values differ when i appears in e. Therefore, a second condition for ensuring equivalence of (2.67) and (2.68) is to rule out textual substitution where e, the replacement term, has a free occurrence of i, a bound variable in the original formula.

We combine these two restrictions to obtain:

(2.69) *Textual Substitution* [*Quantification*]: For i distinct from x and distinct from all variables occurring free in e:

$$(\forall i{:}\, R{:}\, P)^x_e \;=\; (\forall i{:}\, R^x_e{:}\, P^x_e)$$

We can circumvent the restrictions on i, x, and e in Textual Substitution [Quantification] (2.69) by renaming the bound variable to make it distinct from x and every variable occurring free in e. For example, bound variable i in $(\forall i: 0 \le i: i < x)^x_{i+1}$ might first be renamed to j, producing $(\forall j: 0 \le j: j < x)^x_{i+1}$, which does satisfy the restrictions of (2.69) and therefore equals $(\forall j: 0 \le j: j < i+1)$.

Care must be exercised in choosing a new name for a bound variable. Otherwise, the quantified expression that results from the renaming will have a different meaning than the original. The association of bound variable occurrences with quantifiers must be unchanged, and free occurrences of variables must remain unaltered.

We say that a variable v is *captured* when a free occurrence of v becomes a bound occurrence or when the quantifier with which v has been associated changes. For example, renaming i to j in $(\exists i: i < j)$ results in $(\exists j: j < j)$, a quantified expression in which a free occurrence of j has become captured because it became bound. Bound variables can also be captured by a renaming. Renaming i to n in $(\forall i: (\exists n: i \neq n))$ leads to the capture of i by the universal quantifier: $(\forall n: (\exists n: n \neq n))$.

The examples of the preceding paragraph suggest that bound variable renaming be permitted only when the new name does not cause capture:

(2.70) *Bound Variable Renaming Rule*: Provided that

(i) j is different from every free variable that occurs in R and P,

(ii) j is different from every bound variable that occurs in R and P,

then:

$$(Q\, i: R: P) \quad = \quad (Q\, j: R^i_j: P^i_j)$$

Restriction (i) prevents renaming i to j in $(\exists i: i < j)$. Restriction (ii) prevents renaming i to n in $(\forall i: (\exists n: i \neq n))$ or renaming i to a in $(\exists i: i > M)$ for M defined by (2.56).

The need for restrictions (i) and (ii) in Bound Variable Renaming Rule (2.70) is best seen through the rule's soundness proof. We consider the case where "Q" is "$\forall$."

$s \models (\forall j: R^i_j: P^i_j)$
iff «Interpretation for Predicate Logic (2.58)»
For all V: $(s; j{:}\mathrm{V}) \models (R^i_j \Rightarrow P^i_j)$
iff «Textual Substitution [Propositional Connectives] (2.66d)»
For all V: $(s; j{:}\mathrm{V}) \models (R \Rightarrow P)^i_j$
iff «Semantics for Textual Substitution (2.59b)»
For all V: $(s; j{:}\mathrm{V}; i{:}j) \models (R \Rightarrow P)$
iff «$(s; j{:}\mathrm{V}; i{:}j) = (s; j{:}\mathrm{V}; i{:}\mathrm{V})$ due to (2.49) and (2.48)»
For all V: $(s; j{:}\mathrm{V}; i{:}\mathrm{V}) \models (R \Rightarrow P)$
iff «due to (i) and (ii) $R \Rightarrow P$ does not depend on j»

For all V: $(s; i{:}\mathrm{V}) \models (R \Rightarrow P)$
iff «Interpretation for Predicate Logic (2.58)»
$s \models (\forall i: R: P)$

By combining the above laws for textual substitution, we obtain a reasonably straightforward procedure for performing textual substitutions.

(2.71) **Textual Substitution.** For a Predicate Logic formula P, compute P^x_e as follows.

(i) First, use Bound Variable Renaming Rule (2.70) to rename bound variables in P so that they are distinct from each other, from x, and from every variable that occurs free in e.

(ii) Then, use Textual Substitution [Derived Term] (2.63) to replace every derived term Z in P by Z^x_e.

(iii) Finally, use Textual Substitution [Variable] (2.61) to replace every occurrence of variable x in P by e. □

Textual substitution can be generalized to handle simultaneous replacement of more than one variable. This generalization is particularly useful for reasoning about assignment statements. Let $\overline{x}$ denote a list of (not necessarily distinct) variables $x_1, x_2, \ldots, x_n$ and let $\overline{e}$ denote a list of terms $e_1, e_2, \ldots, e_n$, possibly involving the x's. Then for lists $\overline{x}$ and $\overline{e}$, $P^{\overline{x}}_{\overline{e}}$ or $P^{x_1, x_2, \ldots, x_n}_{e_1, e_2, \ldots, e_n}$ denotes the *simultaneous substitution* of $\overline{e}$ for $\overline{x}$ in P.

Simultaneous substitution specifies that variables are replaced only once by terms; unlike nested textual substitutions, substitutions are not repeatedly made into terms that replace the variables. For example, $(x = y)^{x,y}_{y+1,13}$ is $y + 1 = 13$ and not $13 + 1 = 13$. In addition, unlike nested textual substitutions, it is convenient if we define simultaneous substitution to be such that when a variable appears more than once in $\overline{x}$, the rightmost replacement is used. Thus, $(x = y)^{x,x}_{1,2}$ is $2 = y$ and not $1 = y$.

The following definition of simultaneous substitution formalizes these ideas using nested textual substitutions. In it, the apparent reversal in the order of variables is an artifact of our requirement that the rightmost replacement be used for a variable that appears more than once in $\overline{x}$.

(2.72) **Simultaneous Substitution.** Let $y1, y2, \ldots, yn$ be distinct identifiers that do not occur in $\overline{x}$, $\overline{e}$, and P:

$$P^{\overline{x}}_{\overline{e}} \;=\; ((\;\cdots\; ((((\; \cdots\; (P^{xn}_{yn})\; \cdots\;)^{x2}_{y2})^{x1}_{y1})^{yn}_{en})\; \cdots\;)^{y2}_{e2})^{y1}_{e1}$$ □

It is not difficult to prove laws for simultaneous substitution that are analogous to Textual Substitution Laws (2.60) through (2.69). We leave this to the reader.

Inferences with Textual Substitution

Any theorem of Propositional Logic in which propositional variables are replaced by Predicate Logic formulas is a valid formula of Predicate Logic. This is so because theorems of Propositional Logic are *true* for any values of their propositional variables; Predicate Logic formulas substituted for these variables provide one such set of values. We allow Predicate Logic formulas derived in this manner to become theorems of Predicate Logic by having the following inference rules:

(2.73) *Predicate Logic Substitution*: Let $P^{q1, q2, ..., qn}_{Q1, Q2, ..., Qn}$ denote the formula obtained by substituting into Propositional Logic formula P Predicate Logic formulas $Q1, Q2, ..., Qn$ for corresponding propositional variables $q1, q2, ..., qn$. Then:

$$\frac{P}{P^{q1, q2, ..., qn}_{Q1, Q2, ..., Qn}}$$

(2.74) *Predicate Logic Modus Ponens*: Let P and Q be Predicate Logic formulas. Then:

$$\frac{P, P \Rightarrow Q}{Q}$$

We next generalize Substitution of Equals (2.34) to enable individual variables to be replaced.

(2.75) *Substitution of Equals*: For any individual variable x:

$$\text{(a)}\ \frac{Q = R}{P^x_Q = P^x_R} \qquad \text{(b)}\ \frac{Q = R}{P^x_R = P^x_Q}$$

A related formulation, which is sometimes more convenient, is:

(2.76) *Substitution Equivalence Law*: $(e_1 = e_2) \Rightarrow (P^x_{e_1} = P^x_{e_2})$

Substitution Equivalence Law (2.76) asserts that $P^x_{e_1}$ and $P^x_{e_2}$ are equal in states where e_1 equals e_2; in contrast, Substitution of Equals (2.75) only concerns the case where e_1 and e_2 are equal in all states.

The next inference rule allows substitution for a derived term Z, thereby expanding Z according to its definition.

(2.77) *Derived Term Expansion Rule*: For Z a derived term that is well-defined according to Derived Term Restrictions (2.57)

$$Z: \begin{cases} e_1 & \text{if } B_1 \\ \quad \cdots & \\ e_n & \text{if } B_n \end{cases}$$

and P a Predicate Logic formula:

$$\frac{\underset{1 \le k \le n}{\wedge} (B_k = \neg (\underset{j \neq k}{\vee} B_j))}{P^x_Z = ((B_1 \wedge P^x_{e_1}) \ \vee \ \cdots \ \vee \ (B_n \wedge P^x_{e_n}))}$$

We might use this rule to prove that derived term M, defined in (2.56), satisfies $(a < M) = (a < b)$.

$$\begin{array}{ll} & a < M \\ = & \quad \text{«Textual Substitution (2.71)»} \\ & (a < x)^x_M \\ = & \quad \text{«Derived Term Expansion Rule (2.77), since } b < a \ = \ \neg \ (a \le b)\text{»} \\ & (b < a \wedge (a < x)^x_a) \ \vee \ (a \le b \wedge (a < x)^x_b) \\ = & \quad \text{«Textual Substitution (2.71)»} \\ & (b < a \ \wedge a < a) \ \vee \ (a \le b \ \wedge a < b) \\ = & \quad \text{«Arithmetic»} \\ & ((b < a \wedge \mathit{false}) \ \vee \ (a < b)) \\ = & \quad \text{«And-Simplification Law (2.26c)»} \\ & \mathit{false} \ \vee \ a < b \\ = & \quad \text{«Or-Simplification Law (2.25c)»} \\ & a < b \end{array}$$

Laws for Quantification

Textual substitution allows the value of $(\forall x: R: P)$ in a state s to be reformulated as the value in s of an infinite conjunction

$$(2.78) \quad (\forall x: R: P) \ = \ ((R \Rightarrow P)^x_{V_1} \ \wedge \ (R \Rightarrow P)^x_{V_2} \ \wedge \ \cdots \ )$$

where $V_1, V_2, \ldots$ are the constants that any variable (e.g. x) can assume. Similarly, the value of $(\exists x: R: P)$ in a state s can be reformulated in terms of an infinite disjunction:

$$(2.79) \quad (\exists x: R: P) \ = \ ((R \wedge P)^x_{V_1} \ \vee \ (R \wedge P)^x_{V_2} \ \vee \ \cdots \ )$$

These equations, then, enable reasoning informally about quantified expressions by using Propositional Logic.[11] For example, we have the following proof of $(\exists x: \ R: \ P) = \neg \ (\forall x: \ R: \ \neg \ P)$.

$$\begin{array}{ll} & (\exists x: \ R: \ P) \\ = & \quad \text{«Predicate Logic Substitution (2.73) using } (\exists x: \ R: \ P) \\ & \qquad \text{for } p \text{ in Negation Law (2.19)»} \\ & \neg \ (\neg \ (\exists x: \ R: \ P)) \\ = & \quad \text{«(2.79)»} \end{array}$$

[11]Only informal reasoning is possible because (2.78) and (2.79) have ellipses.

$$
\begin{array}{ll}
 & \neg\,(\neg\,((R \wedge P)^x_{V_1} \;\vee\; (R \wedge P)^x_{V_2} \;\vee\; \cdots)) \\
= & \quad \text{«repeated application of De Morgan's Law (2.17b)»} \\
 & \neg\,(\neg\,(R \wedge P)^x_{V_1} \;\wedge\; \neg\,(R \wedge P)^x_{V_2} \;\wedge\; \cdots) \\
= & \quad \text{«Textual Substitution [Propositional Connectives] (2.66b)»} \\
 & \neg\,(\neg\,(R^x_{V_1} \wedge P^x_{V_1}) \;\wedge\; \neg\,(R^x_{V_2} \wedge P^x_{V_2}) \;\wedge\; \cdots) \\
= & \quad \text{«repeated application of De Morgan's Law (2.17a)»} \\
 & \neg\,((\neg\,R^x_{V_1} \vee \neg\,P^x_{V_1}) \;\wedge\; (\neg\,R^x_{V_2} \vee \neg\,P^x_{V_2}) \;\wedge\; \cdots) \\
= & \quad \text{«repeated application of Textual Substitution [Propositional} \\
 & \quad\ \ \text{Connectives] (2.66a) and (2.66c)»} \\
 & \neg\,((\neg\,R \vee \neg\,P)^x_{V_1} \;\wedge\; (\neg\,R \vee \neg\,P)^x_{V_2} \;\wedge\; \cdots) \\
= & \quad \text{«repeated application of Implication Law (2.22a)»} \\
 & \neg\,((R \Rightarrow \neg\,P)^x_{V_1} \;\wedge\; (R \Rightarrow \neg\,P)^x_{V_2} \;\wedge\; \cdots) \\
= & \quad \text{«(2.78)»} \\
 & \neg\,(\forall x:\ R:\ \neg\,P)
\end{array}
$$

A number of other laws can be discovered in this fashion or by further manipulations. Let P, Q, and R be formulas of Predicate Logic. The laws are:

(2.80) *De Morgan's Laws*: (a) $(\exists x:\ R:\ P) \;=\; \neg\,(\forall x:\ R:\ \neg\,P)$
(b) $(\forall x:\ R:\ P) \;=\; \neg\,(\exists x:\ R:\ \neg\,P)$

(2.81) *Conjunction Law*: $(\forall x:\ R:\ P \wedge Q) \;=\; ((\forall x:\ R:\ P) \wedge (\forall x:\ R:\ Q))$

(2.82) *Disjunction Law*: $(\exists x:\ R:\ P \vee Q) \;=\; ((\exists x:\ R:\ P) \vee (\exists x:\ R:\ Q))$

(2.83) *Empty-Range Laws*: (a) $(\forall x:\ \mathit{false}:\ P) \;=\; \mathit{true}$
(b) $(\exists x:\ \mathit{false}:\ P) \;=\; \mathit{false}$

(2.84) *Range Laws*: (a) $(\forall x:\ R \wedge P:\ Q) \;=\; (\forall x:\ R:\ P \Rightarrow Q)$
(b) $(\exists x:\ R \wedge P:\ Q) \;=\; (\exists x:\ R:\ P \wedge Q)$

(2.85) *Range Partitioning Laws*:
(a) $(\forall x:\ P \vee R:\ Q) \;=\; ((\forall x:\ P:\ Q) \wedge (\forall x:\ R:\ Q))$
(b) $(\exists x:\ P \vee R:\ Q) \;=\; ((\exists x:\ P:\ Q) \vee (\exists x:\ R:\ Q))$

(2.86) *Range Narrowing Law*: $(\forall x:\ R:\ P) \;\Rightarrow\; (\forall x:\ R \wedge Q:\ P)$

(2.87) *Range Widening Law*: $(\exists x:\ R:\ P) \;\Rightarrow\; (\exists x:\ R \vee Q:\ P)$

(2.88) *Quantification Weakening Laws*: (a) $(\forall x:\ R:\ P) \;\Rightarrow\; (\forall x:\ R:\ P \vee Q)$
(b) $(\exists x:\ R:\ P) \;\Rightarrow\; (\exists x:\ R:\ P \vee Q)$

(2.89) *Quantification Implication Laws*:
(a) $(\forall x:\ R:\ P \Rightarrow Q) \;\Rightarrow\; ((\forall x:\ R:\ P) \Rightarrow (\forall x:\ R:\ Q))$
(b) $(\forall x:\ R:\ P \Rightarrow Q) \;\Rightarrow\; ((\exists x:\ R:\ P) \Rightarrow (\exists x:\ R:\ Q))$

(2.90) *Distributive Laws*: Provided that x does not occur free in Q:

(a) $(Q \wedge (\exists x\colon R\colon P)) = (\exists x\colon R\colon Q \wedge P)$

(b) $(Q \vee (\forall x\colon R\colon P)) = (\forall x\colon R\colon Q \vee P)$

(c) $(Q \Rightarrow (\forall x\colon R\colon P)) = (\forall x\colon R\colon Q \Rightarrow P)$

(2.91) *Distributive Rules*: Provided that x does not occur free in Q:

(a) $$\frac{(\exists x\colon R)}{(Q \vee (\exists x\colon R\colon P)) = (\exists x\colon R\colon Q \vee P)}$$

(b) $$\frac{(\exists x\colon R)}{(Q \wedge (\forall x\colon R\colon P)) = (\forall x\colon R\colon Q \wedge P)}$$

(c) $$\frac{(\exists x\colon R)}{(Q \Rightarrow (\exists x\colon R\colon P)) = (\exists x\colon R\colon Q \Rightarrow P)}$$

The need for premise $(\exists x\colon R)$ in Distributive Rules (2.91) can be seen by choosing *false* for R and observing that the conclusion of the laws need not be valid.

Here are some laws for introducing or removing quantifiers.

(2.92) *Quantification Simplification Laws*: Provided that x does not occur free in Q:

(a) $(\forall x\colon Q) = Q$

(b) $(\exists x\colon Q) = Q$

(c) $(\forall x\colon R\colon \mathit{true}) = \mathit{true}$

(d) $(\exists x\colon R\colon \mathit{false}) = \mathit{false}$

(2.93) *One-Point Laws*: Provided that x does not occur free in e:

(a) $P^x_e = (\forall x\colon x = e\colon P)$

(b) $P^x_e = (\exists x\colon x = e\colon P)$

The conditions on x in Quantification Simplification Laws (2.92a) and (2.92b) and in One-Point Laws (2.93a) and (2.93b) prevent capture of x.

Next are inference rules that allow introduction and removal of a universal quantifier. If a formula $R \Rightarrow Q$ is valid, then it holds in all states. Thus, $R \Rightarrow Q$ holds for all possible values of any variable, and we have:

(2.94) *Quantification Introduction Rule*: $$\frac{R \Rightarrow Q}{(\forall x\colon R\colon Q)}$$

If, on the other hand, $(\forall x\colon R\colon Q)$ is valid, then Q holds for every value of x that satisfies R. When R^x_e is *false* in a state, $(R \Rightarrow Q)^x_e$ is trivially valid; and when R^x_e is *true* in a state, we conclude from $(\forall x\colon R\colon Q)$ that Q^x_e holds, so $(R \Rightarrow Q)^x_e$ must also be *true* in that state. Thus, whether or not R^x_e holds in a state, we conclude that $(R \Rightarrow Q)^x_e$ holds. This gives:

(2.95) *Quantification Elimination Rule*: $\dfrac{(\forall x\colon R\colon Q)}{(R \Rightarrow Q)^x_e}$

Finally, here are some rules for manipulating formulas involving nested quantifiers. Such nesting can arise when R and P in $(Q\,x : R : P)$ contain quantified expressions.

(2.96) *Quantifier Interchange Laws*: Provided that x does not occur free in Q, and y does not occur free in R:

(a) $(\forall x\colon R\colon (\forall y\colon Q\colon P)) = (\forall y\colon Q\colon (\forall x\colon R\colon P))$

(b) $(\exists x\colon R\colon (\exists y\colon Q\colon P)) = (\exists y\colon Q\colon (\exists x\colon R\colon P))$

(c) $(\exists x\colon R\colon (\forall y\colon Q\colon P)) \Rightarrow (\forall y\colon Q\colon (\exists x\colon R\colon P))$

Note that the converse of Quantifier Interchange Law (2.96c) is not, in general, valid. For example, $(\forall y\colon (\exists x\colon y \in \mathrm{Int} \wedge x \in \mathrm{Rat} \Rightarrow x*y = 1)$ is valid and $(\exists x\colon (\forall y\colon y \in \mathrm{Int} \wedge x \in \mathrm{Rat} \Rightarrow x*y = 1)$ is *false*.

Equational Proofs for Predicate Logic

Equational proofs for Predicate Logic theorems are easier to construct if we can substitute equals for R and P in a quantified expression $(Q\,x : R : P)$. However, Substitution of Equals (2.75) cannot be used for this task when R or P have free occurrences of x, because Textual Substitution (2.71) would rename bound variable x to be distinct from any variables occurring in R and P. For example, using Substitution of Equals (2.75) we would conclude $(\forall x::\ v > 0)^v_{2*x} = (\forall x::\ v > 0)^v_{x+x}$ from premise $2*x = x+x$, but this conclusion is not equivalent to $(\forall x::\ 2*x > 0) = (\forall x::\ x+x > 0)$.

Rules that do allow equals to be substituted for R and P without bound variables being renamed are:

(2.97) *Equals in Quantified Expressions*:

(a) $\dfrac{R = Q}{(\forall x\colon R\colon P) = (\forall x\colon Q\colon P)}$

(b) $\dfrac{R = Q}{(\exists x\colon R\colon P) = (\exists x\colon Q\colon P)}$

(c) $\dfrac{R \Rightarrow (P = Q)}{(\forall x\colon R\colon P) = (\forall x\colon R\colon Q)}$

(d) $\dfrac{R \Rightarrow (P = Q)}{(\exists x\colon R\colon P) = (\exists x\colon R\colon Q)}$

We shall also find it helpful to replace R and P in $(Q\,x : R : P)$ by stronger or weaker formulas. Inference rules for these manipulations subsume Range Narrowing Laws (2.86), Range Widening Law (2.87), Quantification Weakening Laws (2.88), and Quantification Implication Laws (2.89).

(2.98) *Monotonicity in Quantified Expressions*:

$$\text{(a)}\ \frac{R \Rightarrow (P \Rightarrow Q)}{(\exists x:\ R:\ P) \Rightarrow (\exists x:\ R:\ Q)}$$

$$\text{(b)}\ \frac{R \Rightarrow (P \Rightarrow Q)}{(\forall x:\ R:\ P) \Rightarrow (\forall x:\ R:\ Q)}$$

$$\text{(c)}\ \frac{R \Rightarrow (P \Rightarrow Q)}{(\exists x:\ P:\ R) \Rightarrow (\exists x:\ Q:\ R)}$$

$$\text{(d)}\ \frac{\neg R \Rightarrow (P \Rightarrow Q)}{(\forall x:\ Q:\ R) \Rightarrow (\forall x:\ P:\ R)}$$

Notice from Monotonicity in Quantified Expressions (2.98d) that the range of a universally quantified expression behaves as if it has odd parity. The other three inference rules suggest that the body of a universally quantified expression, the range of an existentially quantified expression, and the body of an existentially quantified expression, behave as if they each have even parity.

Given these rules, it often is convenient to give as a justification for a step in an equational proof only the premise for the first in a chain of inferences. For example, we might write

$$\begin{array}{ll} & (\forall x:\ (\forall y:\ (\exists z:\ P))) \\ \Rightarrow & \quad\ \langle\langle P \Rightarrow Q \rangle\rangle \\ & (\forall x:\ (\forall y:\ (\exists z:\ Q))) \end{array}$$

because from premise $P \Rightarrow Q$, Monotonicity in Quantified Expressions (2.98a) derives $(\exists z:\ P) \Rightarrow (\exists z:\ Q)$, and that formula then can be used as a premise for Quantified Expressions (2.98b) to conclude $(\forall y:\ (\exists z:\ P)) \Rightarrow (\forall y:\ (\exists z:\ Q))$, which, in turn, can be used as a premise for Quantified Expressions (2.98b) in order to deduce the desired conclusion:

$$(\forall x:\ (\forall y:\ (\exists z:\ P))) \Rightarrow (\forall x:\ (\forall y:\ (\exists z:\ Q)))$$

For constructing equational proofs, Predicate Logic also has analogues of Propositional Logic inference rules Substitution of Equals (2.34), Monotonicity Rule (2.36), and Antimonotonicity Rule (2.37).

To justify an equational-proof step that asserts $A = B$, we use:

(2.99) *Predicate Logic Substitution of Equals*: For Predicate Logic formulas P^p_Q and P^p_R:

$$\text{(a)}\ \frac{Q = R}{P^p_Q = P^p_R} \qquad\qquad \text{(b)}\ \frac{Q = R}{P^p_R = P^p_Q}$$

In order to formulate Predicate Logic analogues of Monotonicity Rule (2.36) and Antimonotonicity Rule (2.37), we define:

(2.100) **Parity of a Subformula.** For P^p_Q a formula in which subformula Q does not appear in an operand of an equivalence in P:

- subformula Q has *even* parity in P^p_Q iff each occurrence of Q is within an even number of negations, antecedents of implications, and ranges of universal quantifications.
- subformula Q has *odd* parity in P^p_Q iff each occurrence of Q is within an odd number of negations, antecedents of implications, and ranges of universal quantifications. □

Then we have:

(2.101) *Predicate Logic Monotonicity Rule*: For Predicate Logic formulas P^p_Q and R, where subformula Q has even parity in P^p_Q:

$$\frac{Q \Rightarrow R}{P^p_Q \Rightarrow P^p_R}$$

(2.102) *Predicate Logic Antimonotonicity Rule*: For Temporal Logic formulas P^p_Q and R, where subformula Q has odd parity in P^p_Q:

$$\frac{Q \Rightarrow R}{P^p_R \Rightarrow P^p_Q}$$

The conventions concerning justifications for equational proofs in Predicate Logic are summarized by:

(2.103) **Predicate Logic Equational Proof Justifications.** A justification "Why X is a theorem" in an equational proof of a Predicate Logic theorem must be one of the following:

(i) A law or previously proved theorem X of Predicate Logic.

(ii) The theorem of Propositional Logic that enables X to be deduced by Predicate Logic Substitution (2.73). Details of the substitutions may be omitted when they are obvious.

(iii) The theorem of Predicate Logic that enables X to be deduced by Substitution of Equals (2.75), Predicate Logic Substitution of Equals (2.99), Predicate Logic Monotonicity Rule (2.101), or Predicate Logic Antimonotonicity Rule (2.102).

(iv) The theorem that enables X to be deduced by a sequence of one or more other Predicate Logic inference rules. □

2.4 Safety and Liveness Revisited

We can use Predicate Logic and set theory to formalize the notions of property, safety property, and liveness property. These formalizations then allow us to prove that every property is the conjunction of a safety property and a liveness property. Thus, knowledge of how to prove that a program satisfies safety properties and liveness properties suffices for reasoning about any property.

The formalizations require introducing some notation. Let σ be a finite or infinite sequence $s_0\, s_1$ ... of states; the empty sequence is denoted by ε. Define $|\sigma|$ to be the number of states in σ ($|\sigma| = \infty$ if σ is an infinite sequence of states), and also define:

$$\sigma[i]: \quad \begin{cases} s_i & \text{for } i \in \text{Int} \wedge 0 \le i < |\sigma| \\ \varepsilon & \text{otherwise} \end{cases}$$

$$\sigma[..i]: \quad \begin{cases} s_0\, s_1 \ldots s_{\min(i,\, |\sigma|-1)} & \text{for } i \in \text{Int} \wedge 0 \le i \\ \varepsilon & \text{otherwise} \end{cases}$$

$$\sigma[i..]: \quad \begin{cases} s_i\, s_{i+1} \ldots & \text{for } i \in \text{Int} \wedge 0 \le i < |\sigma| \\ \varepsilon & \text{otherwise} \end{cases}$$

An *anchored sequence* is a pair (σ, j), where σ is a finite or infinite sequence of states and j is a natural number that satisfies $j < |\sigma|$. If σ is finite, then (σ, j) is called a *finite anchored sequence*. State sequence σ in anchored sequence (σ, j) is partitioned by j into a sequence $\sigma[0..j-1]$ of *past states*, a *current state* $\sigma[j]$, and a sequence $\sigma[j+1..]$ of *future states*. Thus, not only do anchored sequences describe histories, but they also are expressive enough to describe snapshots of partial executions.

Some sets and operations involving anchored sequences will prove convenient. Let S^+ be the set of nonempty finite-length sequences of program states, and let S^∞ be the set of all finite and infinite sequences of program states (including the empty sequence ε). Define sets of anchored sequences:

$$\Sigma_S^+: \ \{(\sigma, i) \mid \sigma \in S^+ \ \wedge \ i \in \text{Int} \ \wedge \ 0 \le i < |\sigma|\}$$

$$\Sigma_S^\infty: \ \{(\sigma, i) \mid \sigma \in S^\infty \ \wedge \ i \in \text{Int} \ \wedge \ 0 \le i < |\sigma|\}$$

That (σ, i) is a *prefix* of (τ, j) will be denoted by the infix predicate symbol $\le$; it is defined as follows:

$$(\sigma, i) \le (\tau, j): \qquad i \le j \ \wedge \ \sigma[..i] = \tau[..i]$$

For describing certain prefixes of anchored sequences, it is useful to have a notation like the one introduced above for describing prefixes of (ordinary) state sequences:

$$(\sigma, i)[..j]: \qquad (\sigma[..j], \min(i, |\sigma[..j]| - 1))$$

Observe that if (σ, i) is an anchored sequence, then by construction, $(\sigma, i)[..j]$ is also one—no matter what value j has.

The length of an anchored sequence is just the length of its sequence of states:

$$|(\sigma, i)|: \qquad |\sigma|$$

Finally, we define the *catenation* $(\sigma, i)(\tau, j)$ for anchored sequences (σ, i) and (τ, j) as follows, where $\sigma\tau$ is the sequence obtained by appending τ to σ.

$$(\sigma, i)(\tau, j): \qquad (\sigma\tau, i)$$

Formalizing Properties

We model a *property* by a set of anchored sequences. The property Mutual Exclusion, for example, is modeled by a set containing anchored sequences (σ, j) such that all states of σ are ones in which the program counter for at most one process points to an atomic action inside any critical section. The property Termination might be modeled by the set of anchored sequences (σ, j) such that σ is finite and $\sigma[|\sigma| - 1]$ is a state in which the program counter points to the end of the program. Note that σ of an anchored sequence (σ, j) in a property need not correspond to a possible execution of a particular—indeed, any—program.

One way to define a set is by giving a *characteristic predicate*—a predicate that is *true* for members of the set and *false* for other values. Use of a characteristic predicate is attractive because it succinctly describes the shared attributes of a set's elements. In addition, Predicate Logic can be used in reasoning about sets defined with characteristic predicates:

(2.104) *Set Membership*: (a) $\gamma \in \{\sigma \mid P\} \;=\; P^{\sigma}_{\gamma}$
(b) $\gamma \notin \{\sigma \mid P\} \;=\; \neg\, P^{\sigma}_{\gamma}$

We use characteristic predicates for defining properties. The set

$$P: \ \{(\sigma, j) \mid Q\}$$

where

- Q is a predicate whose only free variables are σ and j,
- Q implies that σ is a sequence of states, and
- Q implies that j is an integer satisfying $0 \le j < |\sigma|$

defines P to be the property consisting of all anchored sequences (γ, k) such that $Q^{\sigma,j}_{\gamma,k}$ holds. For example, below we formalize the property that stipulates that the state remains constant.

$$\{(\sigma, j) \mid \sigma \in S^\infty \ \wedge \ j \in \text{Int} \ \wedge \ 0 \le j < |\sigma| \\ \wedge \ (\forall i \in \text{Int}: \ 0 \le i < |\sigma| : \ \sigma[0] = \sigma[i])\}$$

Formalizing Safety and Liveness

Safety (1.1) and Liveness (1.2) of §1.3 are informal definitions. We now formalize them.

Safety (1.1) stipulates that some "bad thing" does not happen. Thus for P to be a safety property, if an anchored sequence α is not in P, then a "bad thing" must occur in some prefix of α. Such a "bad thing" must be irremediable, because a safety property stipulates that the "bad thing" never happens. This suggests that whenever $\alpha \notin P$ holds, there is a finite prefix β that is a "bad thing" for which no γ is a mitigating extension.

(2.105) **Safety Property.** A property P is a safety property iff:

$$(\forall \alpha \in \Sigma^\infty_S : \ \alpha \notin P \ \Rightarrow \ (\exists \beta \in \Sigma^+_S : \ \beta \le \alpha : \ (\forall \gamma \in \Sigma^\infty_S : \ \beta\gamma \notin P))) \qquad \square$$

Note that the only restriction imposed by this definition on the notion of a "bad thing" is that if the "bad thing" happens in α, then there is an identifiable point (i.e., β) at which it occurs.

Mutual Exclusion satisfies (informal definition) Safety (1.1), the "bad thing" being a state in which more than one process is executing in a critical section. To see that Mutual Exclusion also satisfies our formal definition, observe that there is no way to extend a finite anchored sequence that contains such a state to an anchored sequence in which there are no such states.

Liveness (1.2) stipulates that some "good thing" eventually happens. The thing to observe about a liveness property is that no finite anchored sequence is irremediable. This is because if some finite anchored sequence were irremediable, then this would constitute a "bad thing," and a liveness property cannot proscribe a "bad thing" (a safety property does this) but can only prescribe a "good thing." Thus, if P is a liveness property, then for any finite anchored sequence α, there is a β, containing the "good thing," such that $\alpha\beta \in P$ holds.

(2.106) **Liveness Property.** A property P is a liveness property iff:

$$(\forall \alpha \in \Sigma^+_S : \ (\exists \beta \in \Sigma^\infty_S : \ \alpha\beta \in P)) \qquad \square$$

This definition does not restrict what a "good thing" can be. Unlike a "bad thing," a "good thing" can involve an infinite number of states, and it need not occur at an identifiable point.

Recall that Termination satisfies (informal definition) Liveness (1.2), the "good thing" being a state in which the program counter value in the final state of the sequence designates the end of the program. To see that Termination also satisfies our formal definition, for a finite anchored sequence α, choose β to be any finite anchored sequence whose last state is one in which the program counter value is at the end of the program.

Although the "good thing" for Termination occurs at a single identifiable point—the last state—this is not the case for all liveness properties. Consider the (liveness) property that asserts that each process is executed infinitely often. This property contains anchored sequences (σ, j) such that for each process, there is an infinite number of states in σ in which the program counter for that process has changed. Here, a "good thing" is not a finite anchored sequence, and this "good thing" cannot be associated with a single identifiable point in the sequence.

Decomposing Properties into Safety and Liveness

In order to show that every property P can be expressed in terms of safety and liveness properties, we show how to construct a safety property $Safe(P)$ and a liveness property $Live(P)$ such that $P = Safe(P) \cap Live(P)$

$Safe(P)$ contains all anchored sequences in property P as well as those whose prefixes could be extended in a way that would make them part of P.

(2.107) $Safe(P)$: $P \cup \{\delta \mid \delta \in \Sigma_S^\infty \ \wedge\ (\forall \kappa \in \Sigma_S^+ :\ \kappa \le \delta :\ (\exists \iota \in \Sigma_S^\infty :\ \kappa\iota \in P))\}$

To see informally that $Safe(P)$ is a safety property, observe the following. For α an anchored sequence, if $\alpha \notin Safe(P)$ holds, then $\alpha \notin P$ and, by Set Membership (2.104b), $\neg\ (\forall \kappa \in \Sigma_S^+ :\ \kappa \le \delta :\ (\exists \iota \in \Sigma_S^\infty :\ \kappa\iota \in P))_\alpha^\delta$ hold. This means that there exists a finite prefix κ of α that satisfies $(\forall \iota \in \Sigma_S^\infty :\ \kappa\iota \notin P)$, so we could consider κ to be a "bad thing." An anchored sequence that does not satisfy P only because it violates a liveness property will satisfy $Safe(P)$.

To show formally that $Safe(P)$ is a safety property, we show that it satisfies Safety Property (2.105).

1. $\beta \in \Sigma_S^+ \ \wedge\ (\forall \iota \in \Sigma_S^\infty :\ \beta\iota \notin P)$
   $\Rightarrow$ «Range Narrowing Law (2.86)»
   $\beta \in \Sigma_S^+ \ \wedge\ (\forall \iota \in \Sigma_S^\infty :\ \iota = \gamma :\ \beta\iota \notin P) \ \wedge\ (\forall \iota \in \Sigma_S^\infty :\ \beta\iota \notin P)$
   $=$ «One-Point Law (2.93a»
   $\beta \in \Sigma_S^+ \ \wedge\ \beta\gamma \notin P \ \wedge\ (\forall \iota \in \Sigma_S^\infty :\ \beta\iota \notin P)$
   $\Rightarrow$ «Consequent-Weakening Law (2.30)»
   $\beta \in \Sigma_S^+ \ \wedge\ \beta\gamma \notin P \ \wedge\ (\beta\gamma \notin \Sigma_S^\infty \ \vee\ (\forall \iota \in \Sigma_S^\infty :\ \beta\iota \notin P))$
   $=$ «One-Point Law (2.93b)»
   $\beta \in \Sigma_S^+ \ \wedge\ \beta\gamma \notin P \ \wedge\ (\beta\gamma \notin \Sigma_S^\infty \ \vee\ (\exists \kappa :\ \kappa = \beta :\ (\forall \iota \in \Sigma_S^\infty :\ \kappa\iota \notin P)))$
   $\Rightarrow$ «$(\beta \in \Sigma_S^+ \ \wedge\ \kappa = \beta) \ \Rightarrow\ (\kappa \in \Sigma_S^+ \ \wedge\ \kappa \le \beta\gamma)$»
   $\beta\gamma \notin P \ \wedge\ (\beta\gamma \notin \Sigma_S^\infty \ \vee\ (\exists \kappa \in \Sigma_S^+ :\ \kappa \le \beta\gamma :\ (\forall \iota \in \Sigma_S^\infty :\ \kappa\iota \notin P)))$
   $=$ «De Morgan's Laws (2.17) and (2.80)»
   $\beta\gamma \notin P \ \wedge\ \neg\ (\beta\gamma \in \Sigma_S^\infty \ \wedge\ (\forall \kappa \in \Sigma_S^+ :\ \kappa \le \beta\gamma :\ (\exists \iota \in \Sigma_S^\infty :\ \kappa\iota \in P)))$

$=$ «Textual Substitution (2.71)»
$\beta\gamma\notin P \ \wedge\ \neg\,(\delta\in\Sigma_S^\infty \ \wedge\ (\forall\kappa\in\Sigma_S^+:\ \kappa\le\delta:\ (\exists\iota\in\Sigma_S^\infty:\ \kappa\iota\in P)))^{\delta}_{\beta\gamma}$
$=$ «Set Membership (2.104b)»
$\beta\gamma\notin P \ \wedge\ \beta\gamma\notin\{\delta \mid \delta\in\Sigma_S^\infty \ \wedge\ (\forall\kappa\in\Sigma_S^+:\ \kappa\le\delta:\ (\exists\iota\in\Sigma_S^\infty:\ \kappa\iota\in P))\}$
$=$ «definition (2.107) of $Safe(P)$»
$\beta\gamma \notin Safe(P)$

2. $\alpha\in\Sigma_S^\infty \ \wedge\ \alpha\notin Safe(P)$
$=$ «definition (2.107) of $Safe(P)$»
$\alpha\in\Sigma_S^\infty \ \wedge\ \alpha\notin(P \cup \{\delta \mid \delta\in\Sigma_S^\infty \ \wedge\ (\forall\kappa\in\Sigma_S^+:\ \kappa\le\delta:\ (\exists\iota\in\Sigma_S^\infty:\ \kappa\iota\in P))\})$
$\Rightarrow$ «$\alpha\notin(A \cup B) \ \Rightarrow\ \alpha\notin B$»
$\alpha\in\Sigma_S^\infty \ \wedge\ \alpha\notin\{\delta \mid \delta\in\Sigma_S^\infty \ \wedge\ (\forall\kappa\in\Sigma_S^+:\ \kappa\le\delta:\ (\exists\iota\in\Sigma_S^\infty:\ \kappa\iota\in P))\}$
$=$ «Set Membership (2.104b)»
$\alpha\in\Sigma_S^\infty \ \wedge\ \neg\,(\delta\in\Sigma_S^\infty \ \wedge\ (\forall\kappa\in\Sigma_S^+:\ \kappa\le\delta:\ (\exists\iota\in\Sigma_S^\infty:\ \kappa\iota\in P)))^{\delta}_{\alpha}$
$=$ «Textual Substitution (2.71)»
$\alpha\in\Sigma_S^\infty \ \wedge\ \neg\,(\alpha\in\Sigma_S^\infty \ \wedge\ (\forall\kappa\in\Sigma_S^+:\ \kappa\le\alpha:\ (\exists\iota\in\Sigma_S^\infty:\ \kappa\iota\in P)))$
$=$ «De Morgan's Laws (2.17) and (2.80)»
$\alpha\in\Sigma_S^\infty \ \wedge\ (\alpha\notin\Sigma_S^\infty \ \vee\ (\exists\kappa\in\Sigma_S^+:\ \kappa\le\alpha:\ (\forall\iota\in\Sigma_S^\infty:\ \kappa\iota\notin P)))$
$=$ «Implication Law (2.22a)»
$\alpha\in\Sigma_S^\infty \ \wedge\ (\alpha\in\Sigma_S^\infty \Rightarrow (\exists\kappa\in\Sigma_S^+:\ \kappa\le\alpha:\ (\forall\iota\in\Sigma_S^\infty:\ \kappa\iota\notin P)))$
$\Rightarrow$ «Implication-Deduction Law (2.31a)»
$(\exists\kappa\in\Sigma_S^+:\ \kappa\le\alpha:\ (\forall\iota\in\Sigma_S^\infty:\ \kappa\iota\notin P))$
$=$ «Bound Variable Renaming Rule (2.70)»
$(\exists\beta\in\Sigma_S^+:\ \beta\le\alpha:\ (\forall\iota\in\Sigma_S^\infty:\ \beta\iota\notin P))$
$=$ «Quantification Simplification Law (2.92a)»
$(\exists\beta\in\Sigma_S^+:\ \beta\le\alpha:\ (\forall\gamma\in\Sigma_S^\infty:\ (\forall\iota\in\Sigma_S^\infty:\ \beta\iota\notin P)))$
$\Rightarrow$ «line 1»
$(\exists\beta\in\Sigma_S^+:\ \beta\le\alpha:\ (\forall\gamma\in\Sigma_S^\infty:\ \beta\gamma \notin Safe(P)))$

«Quantification Introduction Rule (2.94) with 2»
3. $(\forall\alpha\in\Sigma_S^\infty:\ \alpha\notin Safe(P):\ (\exists\beta\in\Sigma_S^+:\ \beta\le\alpha:\ (\forall\gamma\in\Sigma_S^\infty:\ \beta\gamma\notin Safe(P))))$

4. $(\forall\alpha\in\Sigma_S^\infty:\ \alpha\notin Safe(P):\ (\exists\beta\in\Sigma_S^+:\ \beta\le\alpha:\ (\forall\gamma\in\Sigma_S^\infty:\ \beta\gamma\notin Safe(P))))$
$=$ «Range Law (2.84a)»
$(\forall\alpha\in\Sigma_S^\infty:\ \alpha\notin Safe(P) \ \Rightarrow\ (\exists\beta\in\Sigma_S^+:\ \beta\le\alpha:\ (\forall\gamma\in\Sigma_S^\infty:\ \beta\gamma\notin Safe(P))))$

Since the first line of step 4 is step 3, a theorem, from Equational Proof Conclusions (2.42) we conclude that the final line of step 4 is a theorem. This final line is Safety Property (2.105) instantiated for property $Safe(P)$—we have formally proved that $Safe(P)$ is a safety property.

$Live(P)$ contains all anchored sequences in P as well as those that violate some safety property in P.

(2.108) $Live(P)$: $P \cup \{\delta \mid \delta\in\Sigma_S^\infty \ \wedge\ (\exists\kappa\in\Sigma_S^+:\ \kappa\le\delta:\ (\forall\iota\in\Sigma_S^\infty:\ \kappa\iota \notin P))\}$

An informal justification that $Live(P)$ is a liveness property is the following. All

anchored sequences γ that violate a safety property in P are in $Live(P)$ because by construction, γ will be in $\{\delta \mid \delta \in \Sigma_S^\infty \wedge (\exists \kappa \in \Sigma_S^+ : \kappa \leq \delta : (\forall \iota \in \Sigma_S^\infty : \kappa\iota \notin P))\}$. Therefore every prefix of an anchored sequence in $Live(P)$ can be extended to produce an anchored sequence in $Live(P)$, and according to Liveness Property (2.106), this is the defining characteristic of a liveness property.

To show formally that $Live(P)$ is a liveness property, we prove below:

$$\alpha \in \Sigma_S^+ \Rightarrow (\exists \beta \in \Sigma_S^\infty : \alpha\beta \in Live(P))$$

From this, Quantification Introduction Rule (2.94) yields the desired conclusion, $(\forall \alpha \in \Sigma_S^+ : (\exists \beta \in \Sigma_S^\infty : \alpha\beta \in Live(P)))$.

$$\begin{array}{ll}
 & \alpha \in \Sigma_S^+ \\
= & \quad \text{«And-Simplification Law (2.26b)»} \\
 & \alpha \in \Sigma_S^+ \wedge \mathit{true} \\
= & \quad \text{«Excluded Middle Law (2.20)»} \\
 & \alpha \in \Sigma_S^+ \wedge ((\exists \beta \in \Sigma_S^\infty : \alpha\beta \in P) \vee \neg (\exists \beta \in \Sigma_S^\infty : \alpha\beta \in P)) \\
= & \quad \text{«Bound Variable Renaming Rule (2.70)»} \\
 & \alpha \in \Sigma_S^+ \wedge ((\exists \beta \in \Sigma_S^\infty : \alpha\beta \in P) \vee \neg (\exists \iota \in \Sigma_S^\infty : \alpha\iota \in P)) \\
= & \quad \text{«De Morgan's Laws (2.80b)»} \\
 & \alpha \in \Sigma_S^+ \wedge ((\exists \beta \in \Sigma_S^\infty : \alpha\beta \in P) \vee (\forall \iota \in \Sigma_S^\infty : \alpha\iota \notin P)) \\
= & \quad \text{«Distributive Rule (2.91a), since } (\exists \beta : \beta \in \Sigma_S^\infty)\text{»} \\
 & \alpha \in \Sigma_S^+ \wedge (\exists \beta \in \Sigma_S^\infty : \alpha\beta \in P \vee (\forall \iota \in \Sigma_S^\infty : \alpha\iota \notin P)) \\
= & \quad \text{«Distributive Law (2.90a)»} \\
 & (\exists \beta \in \Sigma_S^\infty : \alpha \in \Sigma_S^+ \wedge (\alpha\beta \in P \vee (\forall \iota \in \Sigma_S^\infty : \alpha\iota \notin P))) \\
\Rightarrow & \quad \text{«}A \wedge (B \vee C) \Rightarrow B \vee (A \wedge C)\text{»} \\
 & (\exists \beta \in \Sigma_S^\infty : \alpha\beta \in P \vee (\alpha \in \Sigma_S^+ \wedge (\forall \iota \in \Sigma_S^\infty : \alpha\iota \notin P))) \\
\Rightarrow & \quad \text{«}\alpha \in \Sigma_S^+ \wedge \beta \in \Sigma_S^\infty \Rightarrow \alpha\beta \in \Sigma_S^\infty\text{»} \\
 & (\exists \beta \in \Sigma_S^\infty : \alpha\beta \in P \vee (\alpha \in \Sigma_S^+ \wedge \alpha\beta \in \Sigma_S^\infty \wedge (\forall \iota \in \Sigma_S^\infty : \alpha\iota \notin P))) \\
= & \quad \text{«One-Point Law (2.93b)»} \\
 & (\exists \beta \in \Sigma_S^\infty : \alpha\beta \in P \vee (\alpha \in \Sigma_S^+ \wedge \alpha\beta \in \Sigma_S^\infty \wedge (\exists \kappa : \kappa = \alpha : (\forall \iota \in \Sigma_S^\infty : \kappa\iota \notin P)))) \\
\Rightarrow & \quad \text{«}(\alpha \in \Sigma_S^+ \wedge \kappa = \alpha) \Rightarrow (\kappa \in \Sigma_S^+ \wedge \kappa \leq \alpha\beta)\text{»} \\
 & (\exists \beta \in \Sigma_S^\infty : \alpha\beta \in P \vee (\alpha\beta \in \Sigma_S^\infty \wedge (\exists \kappa \in \Sigma_S^+ : \kappa \leq \alpha\beta : (\forall \iota \in \Sigma_S^\infty : \kappa\iota \notin P)))) \\
= & \quad \text{«Textual Substitution (2.71)»} \\
 & (\exists \beta \in \Sigma_S^\infty : \alpha\beta \in P \vee (\delta \in \Sigma_S^\infty \wedge (\exists \kappa \in \Sigma_S^+ : \kappa \leq \delta : (\forall \iota \in \Sigma_S^\infty : \kappa\iota \notin P)))^{\delta}_{\alpha\beta}) \\
= & \quad \text{«Set Membership (2.104a)»} \\
 & (\exists \beta \in \Sigma_S^\infty : \alpha\beta \in P \vee \alpha\beta \in \{\delta \mid \delta \in \Sigma_S^\infty \wedge (\exists \kappa \in \Sigma_S^+ : \kappa \leq \delta : (\forall \iota \in \Sigma_S^\infty : \kappa\iota \notin P))\}) \\
= & \quad \text{«}(a \in P \vee a \in Q) = a \in (P \cup Q)\text{»} \\
 & (\exists \beta \in \Sigma_S^\infty : \alpha\beta \in (P \cup \{\delta \mid \delta \in \Sigma_S^\infty \wedge (\exists \kappa \in \Sigma_S^+ : \kappa \leq \delta : (\forall \iota \in \Sigma_S^\infty : \kappa\iota \notin P))\})) \\
= & \quad \text{«definition (2.108) of } Live(P)\text{»} \\
 & (\exists \beta \in \Sigma_S^\infty : \alpha\beta \in Live(P))
\end{array}$$

Finally, we prove that every property can be specified in terms of a safety property and a liveness property.[12]

$$
\begin{array}{ll}
 & \mathit{Safe}(P) \cap \mathit{Live}(P) \\
= & \quad \text{«definition (2.107) of } \mathit{Safe}(P)\text{; definition (2.108) of } \mathit{Live}(P)\text{;} \\
 & \quad TL\colon\ \{\delta \mid (\forall \kappa \in \Sigma_S^\ddagger\colon\ \kappa \le \delta\colon\ (\exists \iota \in \Sigma_S^\infty\colon\ \kappa\iota \in P))\}\text{»} \\
 & (P \cup (\Sigma_S^\infty \cap TL)) \ \cap \ (P \cup (\Sigma_S^\infty \cap \sim TL)) \\
= & \quad \text{«distribution of } \cap \text{ over } \cup\text{»} \\
 & (P \cap P) \ \cup \ (P \cap \Sigma_S^\infty \cap \sim TL) \ \cup \ (\Sigma_S^\infty \cap TL \cap P) \\
 & \quad \cup \ (\Sigma_S^\infty \cap TL \cap \sim TL) \\
= & \quad \text{«}A \cap A = A\text{;} \ \ A \cap \sim A = \varnothing\text{;} \ \ A \cap \varnothing = \varnothing\text{;} \ \ A \cup \varnothing = A\text{»} \\
 & P \ \cup \ (P \cap \Sigma_S^\infty \cap \sim TL) \ \cup \ (\Sigma_S^\infty \cap TL \cap P) \\
= & \quad \text{«}(A \cap B) \subseteq A\text{;} \ \ A \subseteq B \Rightarrow (B \cup A = A)\text{»} \\
 & P \ \cup \ P \ \cup \ P \\
= & \quad \text{«}A \cup A = A\text{»} \\
 & P
\end{array}
$$

Hence, every property P can be partitioned into a safety component $\mathit{Safe}(P)$ and a liveness component $\mathit{Live}(P)$ whose intersection is P.

Historical Notes

An introduction to logic, with particular attention to how it relates to computability and artificial intelligence, can be found in [Hofstadter 79]. The book is well written—it won a Pulitzer Prize—and easy to read.

Two popular texts for a first course in logic are [Church 56] and [Enderton 72]. Church's book takes a proof-theoretic point of view, while Enderton's takes a model-theoretic one. A more advanced text, intended for a first-year graduate course, is [Bell & Machover 77]. The text [Gries & Schneider 93], which was written concurrently with this book, treats at an elementary level all the logics in this chapter. See [DeLong 70] for a comprehensive annotated bibliography for formal logic.

The approach to proofs defined in §2.1 is due to Hilbert, although our equational format was inspired by work of Feijen and first reported in [Dijkstra 85]. An alternative style for presenting proofs is based on the way many people devise proofs, where a theorem is proved by identifying goals that imply the desired result and then proving each goal, perhaps by identifying subgoals, and so on. Logical systems that embody this style of reasoning are called *natural deduction systems*. Gerhard Gentzen developed the first such system in 1935. Textbooks [Quine 61] and [Smullyan 68] describe this approach.

Our presentation in §2.1 of formal logical systems was motivated by [Hofstadter 79], and the PQ–L logic is from there. Our axiomatization of Propositional Logic is from [Church 56], where it is called P_1. The extensions are based on [Gries & Schneider 93]. Many of our rules for quantified expressions are taken from [Hehner 84] and [Dijkstra & Feijen 84].

---

[12] $\sim A$ denotes the complement of set A.

Informal definitions of safety and liveness were first proposed in [Lamport 77a]. The names are borrowed from Petri net theory. The first formal definition of safety did not appear until seven years later [Lamport 85b]. Lamport's definition (Safety Property (2.109) in exercise 2.26) is adequate only for properties that are invariant under stuttering. A formal definition of safety that works for all properties and the first formal definition of liveness are given in [Alpern & Schneider 85]. The first proof that every property is the conjunction of a safety property and a liveness property appeared there as well, although the proof given in §2.4 is based on [Schneider 87]. The relationship between Safety Property (2.109) and Safety Property (2.105) is discussed in [Alpern et al. 86].

Attempts to characterize safety properties and liveness properties in terms of the syntax of the formulas used to express those properties are described in [Sistla 85], [Sistla 86], and [Lichtenstein et al. 85]; automata-theoretic characterizations and an automata-based proof that every property can be decomposed into safety and liveness properties is the subject of [Alpern & Schneider 87]. In [Manna & Pnueli 90], the notions of safety and liveness are further refined into classes based on how that property can be proved. See [Kindler 94] for a survey of research concerned with defining safety and liveness properties.

Exercises

2.1. (a) Give a finite set of axioms that can be added to PQ–L to make it sound and complete under Addition-Equality Interpretation (2.4).

(b) Prove that the new logical system is sound and complete. (Hint: To show that a logic is complete, use induction, where the induction hypothesis is that every valid formula with fewer than i symbols is provable.)

2.2. Consider the formal logical system defined by

Symbols: M, I, o.

Formulas: Formulas have the form $a\ M\ b\ I\ c$ where each of a, b, and c is a sequence of zero or more o's.

Axiom: $A1.\ \ o\ o\ M\ o\ o\ I\ o\ o\ o\ o$

Inference Rules: For a, b, and c each a sequence of zero or more o's:

$$R1:\ \ \frac{a\ M\ b\ I\ c}{a\ a\ M\ b\ I\ c\ c}$$

$$R2:\ \ \frac{a\ M\ b\ b\ I\ c\ c}{a\ a\ M\ b\ I\ c\ c}$$

(a) Give five formulas in the logic.

(b) State and prove five theorems of the logic.

(c) Give an interpretation of the logic that makes multiplication of integers a model.

(d) Give a formula that is *true* according to your interpretation but is not a theorem. Give additional axioms and/or inference rules to make the logic complete.

2.3. Two possible definitions for soundness of an inference rule are:

Theorem Soundness. An inference rule is *sound* if a formula derived using that rule is valid whenever the premises used are theorems.

Model Soundness. An inference rule is *sound* if a formula derived using that rule is valid whenever the premises used in that inference are valid.

What are the advantages/disadvantages of axiomatizations in which all inference rules satisfy Theorem Soundness versus Model Soundness?

2.4. Translate the following English statements into Propositional Logic, assuming:

Prime_n: "*n* is prime"

Odd_n: "*n* is odd"

Even_n: "*n* is even"

(a) If *n* is not even then it is odd.

(b) If *n* is prime then it is odd, in which case it is not even.

(c) If *n* is even and prime then it is also odd.

2.5. Define appropriate propositions and then formulate the following statements as formulas of Propositional Logic.

(a) If I am here then I am not there.

(b) I cannot be here and there at the same time.

(c) I cannot be here unless I am not there.

(d) Whenever the program terminates, it produces the correct answer.

(e) The program produces the correct answer only if it terminates.

(f) The program produces the correct answer or it does not terminate.

2.6. The formula $P_1 \oplus P_2 \oplus ... \oplus P_n$ is intended to be *true* when exactly one of $P_1, P_2, ..., P_n$ is. Show how to translate $P_1 \oplus P_2 \oplus ... \oplus P_n$ into Propositional Logic.

2.7. Give proofs using Propositional Logic for the following theorems.

(a) $((p \wedge q) \Rightarrow r) \Rightarrow (p \Rightarrow (q \Rightarrow r))$

(b) $(((p \Rightarrow q) \wedge (q \Rightarrow \mathit{false})) \Rightarrow (p \Rightarrow \mathit{false}))$

(c) $((p \Rightarrow q) \wedge (p \Rightarrow r)) \Rightarrow (p \Rightarrow (q \wedge r))$

(d) $(p \Rightarrow (p \Rightarrow \mathit{false})) \Rightarrow (p \Rightarrow \mathit{false})$

(e) $(p \Rightarrow (q \Rightarrow \mathit{false})) \Rightarrow (q \Rightarrow (p \Rightarrow \mathit{false}))$

(f) $(p \Rightarrow q) \Rightarrow ((q \Rightarrow \mathit{false}) \Rightarrow (p \Rightarrow \mathit{false}))$

2.8. Compute the value of each of the following Propositional Logic formulas in every possible state.

(a) $p \vee q \vee \neg p$

(b) $p \vee (p \Rightarrow q)$

(c) $\neg (p \Rightarrow q) \Rightarrow (p \wedge (q \Rightarrow \mathit{false}))$

(d) $(p \vee q) \wedge (\neg p \vee q)$

(e) $(p \wedge q) \vee (\neg p \wedge q)$

(f) $\neg (\neg p \wedge (q \Rightarrow p)) \vee \neg q$

2.9. Replace axioms (2.7)–(2.9) in the axiomatization of Propositional Logic with the following four axioms:

A1. $(p \Rightarrow q) \Rightarrow ((q \Rightarrow r) \Rightarrow (p \Rightarrow r))$

A2. $(((p \Rightarrow q) \Rightarrow p) \Rightarrow p)$

A3. $p \Rightarrow (q \Rightarrow p)$

A4. $false \Rightarrow p$

Show that every theorem of this new axiomatization is also a theorem of the original and vice versa.

2.10. A *contradiction* in Propositional Logic is a formula that is *false* in every state.

(a) Prove, using Propositional Logic, that if P is a theorem then $P \Rightarrow false$ is a contradiction.

(b) Prove, using Propositional Logic, that if P is a theorem then $\neg P$ is a contradiction.

2.11. Suppose we add $p \Rightarrow false$ as an axiom to our axiomatization of Propositional Logic. Show that whenever P is a theorem of the resulting logic, so is $P \Rightarrow false$. What is the consequence of this with respect to the utility of the new logic?

2.12. A formula of Propositional Logic is in *conjunctive normal form* if it is a conjunction of formulas, each of which is the disjunction of propositional variables and/or their negations. Show, using Extended Propositional Logic, that it is always possible to translate a formula into conjunctive normal form.

2.13. A formula of Propositional Logic is in *disjunctive normal form* if it is a disjunction of formulas, each of which is the conjunction of propositional variables and/or their negations. Show, using Extended Propositional Logic, that it is always possible to translate a formula into disjunctive normal form.

2.14. Use the equational proof format and simplify the following formulas.

(a) $p \vee (q \vee p) \vee \neg q$

(b) $p \wedge (q \Rightarrow p) \wedge (p \vee \neg q)$

(c) $\neg p \Rightarrow (\neg p \wedge q)$

(d) $(p \wedge (q \Rightarrow false)) \;=\; \neg (p \Rightarrow \neg p)$

(e) $p \Rightarrow (q \Rightarrow (p \wedge q))$

(f) $(p \wedge q \wedge r) \vee (\neg p) \vee (\neg q) \vee (\neg r)$

(g) $p \wedge q \Rightarrow (\neg p \wedge \neg q)$

(h) $p \wedge q \wedge (p \Rightarrow r) \Rightarrow (r \vee (q \Rightarrow r))$

(i) $p \Rightarrow ((r \vee s) \Rightarrow p)$

(j) $q \Rightarrow (r \Rightarrow (q \wedge r))$

(k) $((p \Rightarrow q) \Rightarrow p) \Rightarrow ((p \Rightarrow false) \Rightarrow p)$

2.15. Show how the effects of inference rules Substitution of Equals (2.34), Transitivity of Equality (2.38), and Transitivity of Implication (2.39) could be achieved using only axioms (2.7), (2.8), and (2.9) and inference rules Modus Ponens (2.10) and Substitution (2.11) of Propositional Logic.

2.16. Show how it is always possible to prove P a theorem of Propositional Logic whenever $(P \Rightarrow true) \wedge (true \Rightarrow P)$ is a theorem by giving a method to extend a proof for $(P \Rightarrow true) \wedge (true \Rightarrow P)$ into one for P.

2.17. Formulate the following as formulas of Predicate Logic, assuming that Obj is the set of things, Plc is the set of places, and:

$yech(t)$: "thing t is rotten"

$loc(t, p)$: "thing t is located at place p"

(a) Everything is rotten.

(b) Something is rotten.

(c) Nothing is rotten.

(d) Something is rotten in Denmark.

(e) Nothing in Denmark is rotten, but something someplace else is.

(f) Something is rotten someplace and nothing is rotten everywhere.

2.18. Write Predicate Logic formulas for the following statements. Assume that $a[1 .. n]$ and $b[1 .. m]$ are arrays of integers.

(a) All elements in a are nonzero.

(b) Some element in a is zero.

(c) Array a is sorted in ascending order.

(d) Every element in a is also in b.

(e) No value in a appears two or more times in b.

(f) The sequence of elements $a[1], a[2], ..., a[n]$ forms a palindrome. (A *palindrome* is a sequence of characters that reads the same both forwards and backwards, e.g., 1234321.)

2.19. Where possible, perform the indicated substitutions into:

$$E\colon\ x < y\ \wedge\ (\forall i\colon\ 0 \le i < n\colon\ b[i] < y)$$

(a) E^x_z (b) E^x_{x+y} (c) E^i_j (d) $(E^y_{w*z})^y_{a+u}$

(e) $E^{y,y}_{w*z,a+u}$ (f) $(E^y_{w*z})^z_{a+u}$ (g) $E^{y,z}_{w*z,a+u}$

2.20. Use Propositional Logic, informal meaning (2.78) of a universally quantified expression, and informal meaning (2.79) of an existentially quantified expression to justify the following equivalences.

(a) Conjunction Law (2.81)

(b) Disjunction Law (2.82)

(c) Empty-Range Law (2.83a)

(d) Empty-Range Law (2.83b)

(e) Range Law (2.84a)

(f) Range Law (2.84b)

(g) Range Partitioning Law (2.85a)

(h) Range Partitioning Law (2.85b)

(i) Range Narrowing Law (2.86)

(j) Range Widening Law (2.87)

(k) Quantification Weakening Law (2.88a)

(l) Quantification Weakening Law (2.88b)

2.21. Simplify the following formulas.

(a) $(\forall i:\ 1 \le i < n:\ r < A[i]) \ \wedge \ r < A[n]$

(b) $(\forall i:\ 1 \le i \le 100:\ 1 \le i \le 100 \Rightarrow A[i] > 7)$

(c) $(\forall k:\ false:\ k \neq k) \Rightarrow k = k$

(d) $(\exists k:\ false:\ k \neq k) \Rightarrow k = k$

(e) $(\forall x:\ R:\ P) \Rightarrow (\exists x:\ R:\ P)$

(f) $(\forall x:\ R:\ P) \Rightarrow (\exists i:\ P:\ R)$

2.22. Prove the following theorems of Predicate Logic.

(a) $x = e \ \Rightarrow \ P = P^x_e$

(b) $(\forall x:\ R:\ Q) \Rightarrow (R \wedge Q)^x_e$

(c) $(\exists x:\ R:\ P) \vee (\exists x:\ R:\ \neg P)$

(d) $\neg ((\forall x:\ R:\ P) \wedge (\forall x:\ R:\ P))$

(e) $(\forall x:\ P) \Rightarrow (\exists x:\ x = 22:\ P)$

(f) $(\forall x:\ R:\ Q) \Rightarrow (\forall x:\ R:\ P \Rightarrow Q)$

2.23. Show that Equals in Quantified Expressions (2.97a)–(2.97d) can be replaced by a single rule:

$$\frac{P \ = \ Q}{(\forall x:\ P) \ = \ (\forall x:\ Q)}$$

2.24. Under what conditions does $(E^{\overline{x}}_{\overline{u}})^{\overline{u}}_{\overline{x}} = E$ hold?

2.25. Give either a proof or a counterexample for each of the following claims.

(a) The intersection of two safety properties is always a safety property.

(b) The union of two safety properties is always a safety property.

(c) The complement of a safety property is always a safety property.

(d) The intersection of two liveness properties is always a liveness property.

(e) The union of two liveness properties is always a liveness property.

(f) The complement of a liveness property is always a liveness property.

(g) The union of any nonempty property and a liveness property is a liveness property.

(h) Every nontrivial property is the intersection of two nontrivial liveness properties.

2.26. Consider the following formal definition for a safety property P

(2.109) **Safety Property:** $(\forall \alpha \in \Sigma_S^\infty:$
$\alpha \in P = (\forall (\beta, j) \in \Sigma_S^+: (\beta, j) \leq \alpha: (\beta, j)(\beta[\,|\beta|-1]^\omega, k)) \in P))$

where $\beta[\,|\beta|-1]^\omega$ denotes an infinite sequence of $\beta[\,|\beta|-1]$'s.

(a) Give an example of a property that satisfies Safety Property (2.105) but not Safety Property (2.109).

(b) A property P is *invariant under stuttering* iff $\gamma \in P$ then $\delta \in P$ and vice versa, where δ is γ with every state repeated zero or more times. Prove that Safety Property (2.109) is the same as Safety Property (2.105) for properties that are invariant under stuttering.

2.27. Prove the following:

(a) P is a safety property iff $P = \mathit{Safe}(P)$ holds.

(b) P is a liveness property iff $P = \mathit{Live}(P)$ holds.

2.28. Suppose that properties are sets of finite and infinite sequences of states (rather than sets of anchored sequences).

(a) What would the formal definition be for safety properties?

(b) What would the formal definition be for liveness properties?

(c) Give definitions for $\mathit{Safe}(P)$ and $\mathit{Live}(P)$ and prove that for a property P:

$\mathit{Safe}(P)$ is a safety property.

$\mathit{Live}(P)$ is a liveness property.

$P = \mathit{Safe}(P) \cap \mathit{Live}(P)$.