A Programmer's Introduction to Predicate Logic

Technical Report UMCIS–1994–02

H. Conrad Cunningham
cunningham@cs.olemiss.edu

Software Architecure Research Group
Department of Computer and Information Science
University of Mississippi
203 Weir Hall
University, Mississippi 38677 USA

February 1994

Revised January 1996
Reprinted January 2006

H. Conrad Cunningham, D.Sc.
Associate Professor
Department of Computer and Information Science
University of Mississippi
203 Weir Hall
University, Mississippi 38677
USA

cunningham@cs.olemiss.edu

Acknowledgements

The preparation of this document was supported by the National Science Foundation under Grant CCR-9210342 and by the Department of Computer and Information Science at the University of Mississippi.

I thank Jerome Y. Plun for his detailed reading of this report and suggestions for improvements.

PRELIMINARIES

References

This document seeks to be compatible with Edward Cohen's textbook *Programming in the 1990's: An Introduction to the Calculation of Programs* (Springer-Verlag, 1990). Some of the ideas and presentation are also influenced by David Gries and Fred B. Schneider's textbook *A Logical Approach to Discrete Math* (Springer-Verlag, 1993) and other sources.

Expressions

An *expression* is a textual entity consisting of symbols that represent constant values, variables, and mathematical operations.

A *state* is a mapping of the variables to their respective values.

A *state space* is the set of all possible states.

Evaluation of an expression in a state is performed by replacing all the variables in the expression by their respective values and then computing the resulting value.

Textual Substitution

Let E and F be expressions and x be a variable. The notation

$$E^x_F \qquad \text{or} \qquad E(x := F)$$

represents an expression that is the same as E except that all occurrences of variable x are replaced by "(F)". Any unneeded parentheses may be removed.

This operation is called *textual substitution*. It binds more tightly than any other operation. Thus, in general, parentheses will need to be used to show the scope of the substitution.

Examples, where $E = x + y \leq z$:

$$\begin{array}{lcl}
E(z := 4) & = & x + y \leq 4 \\
E(n := 4) & = & x + y \leq z \\
E(z := z) & = & x + y \leq z \\
E^x_{x+1} & = & (x+1) + y \leq z \\
E^y_{y-x} & = & x + (y - x) \leq z \\
(E^y_{x+2})^x_z & = & z + (z + 2) \leq z \\
(x * y + z)(y := x + k) & = & x * (x + k) + z \\
2 * (x * y + z)(y := x + k) + 3 & = & 2 * (x * (x + k) + z) + 3 \\
E^{x+y}_{y+x} & & \text{is undefined!}
\end{array}$$

This notation can be extended to allow textual substitution for several variables *simultaneously*.

Examples, again where $E = x + y \leq z$:

$$\begin{array}{lcl} E^{x,y}_{x+2,1} & = & (x+2)+1 \leq z \\ E^{x,y}_{y,x} & = & y + x \leq z \\ E(x, z := z, 4) & = & z + y \leq 4 \\ (E^{x,y}_{y,x})^{x,y}_{y,x} & = & x + y \leq z \end{array}$$

Equality of Expressions

Evaluation of the expression $X = Y$ in a state yields the Boolean value TRUE if expressions X and Y have the same value and yields FALSE otherwise. This defines *equality* in terms of expression evaluation (i.e., semantically).

Equality can also be defined in terms of how expressions involving "=" can be manipulated (i.e., syntactically). We give four axioms.

Reflexivity: $x = x$

Symmetry (commutativity): $(x = y) = (y = x)$

Transitivity: $\dfrac{X = Y,\ Y = Z}{X = Z}$

This is an *inference rule*. The expressions above the line are called *premises* or *hypotheses*; the expressions below the line are called *conclusions*. An inference rule asserts that if the premises are taken to be truths, then the conclusions are also truths.

The Transitivity rule means that, from hypotheses $X = Y$ and $Y = Z$, we can conclude $X = Z$.

Leibniz: $\dfrac{X = Y}{E(z := X) \ = \ E(z := Y)}$

That is, "equals can be replaced by equals". (Be careful! You must not forget the implicit parentheses that you have in an expression.)

Functions

A function is a rule for computing a value, say v, from another value, say w. Value w is called the *argument* and value v is the corresponding *result*.

For some function f and argument w, expression $f.w$ denotes the *application* of function f to value w. Note that we are using the symbol "." as an explicit function application operator. That is, we write $f.w$ instead of the more conventional $f(w)$.

The function application operator "." has higher binding power than any operator except textual substitution. That is, for any such operator $\odot$, $f.x \odot y = (f.x) \odot y$.

The function application operator binds to the left. That is, $f.x.y = (f.x).y$.

Often we will define a function f with argument x and defining expression $5 * x + 13$ as follows:

$$f.x : 5 * x + 13$$

(We use "$*$" to denote multiplication.)

For some expression E, if we define

$$f.x : E$$

then, for any expression Z,

$$f.Z = E(x := Z)$$

thus defining function application in terms of textual substitution. We can thus reformulate the Leibniz rule as follows:

Leibniz: $$\frac{X = Y}{f.X = f.Y}$$

Binary Relational Operators

We use the standard relational operators. However, we pronounce them in a way that avoids negative terminology.

Operator Symbol	**Pronunciation**
$=$	“equals”
$\neq$	“differs from”
$<$	“less than”
$\leq$	“at most”
$\geq$	“at least”
$>$	“greater than”

As a convenience, we also allow use of these binary operators in a *conjunctional* manner. That is, the expression $0 \leq x < 10$ is true if and only if both $0 \leq x$ and $x < 10$ are true.

Natural Numbers

Mathematicians normally consider the set of natural numbers to be the set $\{1, 2, 3, \cdots\}$.

Computing scientists normally consider the natural numbers to be the set $\{0, 1, 2, \cdots\}$.

Why include 0? Zero of something often occurs in the physical world and in computing science. What is the speed of an object at rest? What is the address offset from the beginning of an array to the first element of the array?

The inclusion of 0 in the set of natural numbers thus helps avoid unnecessary case analyses in our programs and their proofs.

PREDICATE LOGIC

Predicates

A *predicate* is an expression representing a function from a domain to the set { TRUE, FALSE }, (i.e., a Boolean function). Our domain of interest is the *state space* for a program, the Cartesian product of the type sets for the program's variables. Unless otherwise stated, we will assume that all predicates are total Boolean functions on the state space.

Syntax

A predicate is an expression formed from primitive predicates using parentheses and the following Boolean operators and quantifiers.

Operation	**Symbol**	**Pronunciation**	**Binding Power**
equivalence	$\equiv$	"equivales"	1
disjunction	$\vee$	"or"	3
conjunction	$\wedge$	"and"	3
implication	$\Rightarrow$	"implies"	2
consequence	$\Leftarrow$	"follows from"	2
negation	$\neg$	"not"	4
discrepancy	$\not\equiv$	"differs from"	1
universal quantification	$\forall$	"for all"	
existential quantification	$\exists$	"there exists"	

Predicate expressions are formed according to the following rules:

1. The primitive predicates include *true*, *false*, and the usual relational expressions from arithmetic, e.g., $x^2 + y^2 = 1$ and $x \leq 1$. (Use the usual parenthesis conventions for the arithmetic expressions. The arithmetic and relational operators have higher binding power than the Boolean operators.)

2. If P and Q are predicates, then $(P \equiv Q)$, $(P \vee Q)$, $(P \wedge Q)$, $(P \Rightarrow Q)$, $(P \Leftarrow Q)$, and $(P \not\equiv Q)$ are predicates.

3. If P is a predicate, then $(\neg P)$ and (P) are predicates.

4. If R and T are predicates and x is not a program variable, then $(\forall\, x : R : T)$ and $(\exists\, x : R : T)$ are predicates.

   - Predicate R is called the *range*, predicate T the *term*, and x the *dummy* variable.
   - Dummy variable x is *bound* by the *quantifier* and, hence, is sometimes called a bound or quantified variable. Variables which are not bound are sometimes said to be *free*. More than one variable may be bound by one quantifier, e.g., $(\forall\, x, y : R : T)$.
   - If the range is the predicate *true*, then it may be left blank, e.g., $(\forall\, x :: T)$.

Predicate expressions may be simplified by omitting parentheses according the following rules:

1. The parentheses around universal and existential quantifications cannot be omitted. Otherwise, the parenthesis around an entire predicate expression may be omitted.

2. The outermost parentheses in subexpressions $(\neg P)$ and $((P))$ may be omitted.

3. If Boolean operator $\otimes$ has a *higher binding power* than operator $\oplus$, then the parentheses in subexpressions $(P \otimes Q) \oplus R$ and $R \oplus (P \otimes Q)$ may be omitted.

4. If Boolean operators $\otimes$ and $\odot$ have the same binding power and associate freely with each other, then the parentheses in subexpressions $(P \otimes Q) \odot R$ and $P \otimes (Q \odot R)$ may be omitted.

Semantics

For predicates P and Q, the Boolean values of the following predicate expressions are assigned as follows. A point in the state space corresponds to a unique "vector" of values for the variables.

1. The expression *true* is TRUE at every point of the state space.

2. The expression *false* is FALSE at every point of the state space.

3. An arithmetic relation predicate is TRUE at every point of the state space in which the expression resulting from the substitution of the values (at that point) for the variables (in the expression) evaluates to TRUE in the arithmetic; it is FALSE elsewhere.

4. $P \equiv Q$ is TRUE at every point of the state space where the values of P and Q are equal; it is FALSE elsewhere.

5. $P \vee Q$ is FALSE at every point of the state space where both P and Q are FALSE;
it is TRUE elsewhere.

6. $P \wedge Q$ is TRUE at every point of the state space where both P and Q are TRUE; it is FALSE elsewhere.

7. $P \Rightarrow Q$ is FALSE at every point of the state space where P is TRUE and Q is FALSE; it is TRUE elsewhere.

8. $P \Leftarrow Q$ is FALSE at every point of the state space where P is FALSE and Q is TRUE; it is TRUE elsewhere.

9. $\neg P$ is TRUE at every point of the state space where P is FALSE; it is FALSE elsewhere.

10. $P \not\equiv Q$ is TRUE at every point of the state space where the values of P and Q differ; it is FALSE elsewhere.

11. $(\forall\, x : P : Q)$ is TRUE at every point of the state space in which, for any value v,
$P(x := v) \Rightarrow Q(x := v)$ is TRUE; it is FALSE elsewhere.

12. $(\exists\, x : P : Q)$ is TRUE at every point of the state space in which, for some value v,
$P(x := v) \wedge Q(x := v)$ is TRUE; it is FALSE elsewhere.

Equivalence versus Equality

If P and Q are predicates, then $P = Q$ has the same value as $P \equiv Q$.

Why have "$\equiv$" when "$=$" gives the same result?

Because we want to *use* "$=$" and "$\equiv$" in different ways in expressions.

First, we can give the binary Boolean operators lower binding powers than the binary arithmetic operators, thus eliminating the need for parentheses in many cases. For example, we can write $x * y = 0 \equiv x = 0 \vee y = 0$ instead of $(x * y = 0) = (x = 0 \vee y = 0)$.

Second, we can make $=$ *conjunctional* and $\equiv$ *associative*. For example, we can write $P = Q = R$ as a convenient alternative for $(P = Q) \wedge (Q = R)$ and write $P \equiv Q \equiv R$ as alternative for $(P \equiv Q) \equiv R$ and $P \equiv (Q \equiv R)$.

Validity

A predicate that evaluates to TRUE at every point in the state space is especially useful for reasoning. Such a predicate is said to be *valid*.

Note that a predicate P is valid if and only if the predicate $P \equiv true$ is valid.

If P is valid, we also write P *holds* or, when the context is clear, just P.

Several valid predicates (i.e., "laws") appear in the tables at the end of this document. In these laws, the symbols P, Q, and R denote arbitrary predicate expressions. We can show that these laws are valid by using the evaluation rules (i.e., the semantics) that are given above.

Semantic Reasoning (Model)

We can prove the validity of a predicate expression by showing that, for an arbitrary point in the state space, any combination of values for the variables that make up the expression will result in a value of TRUE for the expression. We can sometimes use a *truth table* to do this.

Consider $P \vee Q \equiv Q \vee P$, the symmetry (commutativity) of the $\vee$. (We give this as an axiom later.)

The table below uses T for TRUE and F for FALSE. Note that the values under the $\equiv$ heading are all TRUE.

P	Q	$P \vee Q$	$\equiv$	$Q \vee P$
T	T	T	**T**	T
T	F	T	**T**	T
F	T	T	**T**	T
F	F	F	**T**	F

Syntactic Reasoning

Although we can prove the validity of a predicate by evaluating the predicate expression, it is usually more convenient to reason about the predicate syntactically (by manipulating symbols) rather than semantically (by evaluating the expression). That is, we want a *calculus* of predicates.

For our predicate calculus, we choose some of the laws as *axioms* (i.e., postulates) which we assume to hold. (We can, of course, justify the choice of an axiom by evaluating the expression and showing that it is valid.)

Other laws are *theorems* which we can prove to be valid by using the previous axioms and theorems, the Transitivity and Leibniz inference rules given above, and the Substitution inference rule given below. (Of course, an axiom is a special case of a theorem.)

Substitution: $\dfrac{E}{E(z := F)}$

We interpret this inference rule to mean that, if expression E holds, then $E(z := F)$ also holds.

Equivalence Axioms and Proof Methods

To begin the calculus, we postulate that $\equiv$ is associative and symmetric (i.e., commutative) and has the identity element *true*. Remember that P and Q denote arbitrary predicate expressions.

Associativity: $((P \equiv Q) \equiv R) \equiv (P \equiv (Q \equiv R))$

Symmetry: $(P \equiv Q) \equiv (Q \equiv P)$

Identity: $P \equiv \mathit{true} \equiv P$

Note that the associativity axiom allows us to rewrite the identity axiom as either
$(P \equiv \mathit{true}) \equiv P$ or $P \equiv (\mathit{true} \equiv P)$.

Now we can prove our first theorem $P \equiv P$, reflexivity.

Proof Method: To prove X is a theorem, transform X into an axiom or a previously proved theorem using the Leibniz inference rule.

$P \equiv P$

$\equiv$ $\langle$ $\equiv$ identity written by associativity as $P \equiv (\mathit{true} \equiv P)$,
Leibniz justifies textual substitution for second P $\rangle$

$P \equiv (\mathit{true} \equiv P)$

i.e., $\equiv$ identity axiom

The text between the angle-brackets gives hints why the predicate on the line above it equals the predicate on the line below it. These hints usually describe the premises that enable us to use the Leibniz inference rule to conclude that the two predicates are equal . Since every step involves Leibniz, we normally do not state it explicitly as a justification for the step. The Transitivity inference rule allows us to conclude that the first predicate in the sequence of steps equals the last predicate in the sequence.

Note that Leibniz and the hints justify the transformation in either direction—from the line above to the line below or vice versa.

Now a second theorem: *true*.

$$\begin{array}{ll} & true \\ \equiv & \quad \langle \equiv \text{ identity written } true \equiv (true \equiv true) \text{ by associativity and substitution of } true \text{ for } P \ \rangle \\ & true \equiv true \\ & \quad \text{i.e., } \equiv \text{ reflexivity theorem} \end{array}$$

Note: The Substitution inference rule allows us to conclude the predicate $true \equiv (true \equiv true)$ from the axiom scheme $P \equiv (true \equiv P)$ by substituting $P := true$.

Observation: Since $\equiv$ is symmetric and has identity *true*, we can replace any occurrence of $P \equiv true$ or $true \equiv P$ by just P without changing the value of the expression. We usually make such replacements without giving explicit hints.

The following "metatheorem" is often useful. We do not prove it here.

Metatheorem: If P and Q are theorems, then $P \equiv Q$ is also a theorem.

Disjunction Axioms and More Proof Methods

One way to prove theorems of the form $X \equiv Y$ is to transform $X \equiv Y$ into *true* (or some other axiom or previously proved theorem). Another, often more concise and elegant, method is the following.

Proof Method: To prove that $X \equiv Y$ is a theorem, transform X to Y or Y to X using the Leibniz inference rule.

To justify this method formally, note that the steps needed to prove that X equivales Y are the same steps needed to prove that $X \equiv X$ equivales $X \equiv Y$. Thus a proof of $X \equiv Y$ can be mechanically translated to a proof of $(X \equiv X) \equiv (X \equiv Y)$.

Now we introduce the disjunction operator $\vee$ by means of four axioms. Disjunction has higher binding power than $\equiv$.

Associativity: $(P \vee Q) \vee R \equiv P \vee (Q \vee R)$

Symmetry: $P \vee Q \equiv Q \vee P$

Idempotence: $P \vee P \equiv P$

Distribution of $\vee$ over $\equiv$: $P \vee (Q \equiv R) \equiv P \vee Q \equiv P \vee R$

Now let's prove the distribution of $\vee$ over itself: $P \vee (Q \vee R) \equiv (P \vee Q) \vee (P \vee R)$

We seek to prove this equivalence by showing that one side can be transformed into the other. We use the following heuristic (from Gries/Schneider) for developing the proof.

Heuristic: Identify applicable laws by matching the syntactic structure of expressions or subexpressions. The operators that appear in a predicate expression and the shape of its subexpressions can focus the choice of laws to be used in manipulating it.

$$\begin{array}{ll}
 & (P \vee Q) \vee (P \vee R) \\
\equiv & \quad \langle \vee \text{ associativity } \rangle \\
 & P \vee Q \vee P \vee R \\
\equiv & \quad \langle \vee \text{ symmetry } \rangle \\
 & P \vee P \vee Q \vee R \\
\equiv & \quad \langle \vee \text{ idempotence } \rangle \\
 & P \vee Q \vee R \\
\equiv & \quad \langle \vee \text{ associativity } \rangle \\
 & P \vee (Q \vee R)
\end{array}$$

We often will leave appeals to associativity and symmetry unstated. This allows us to rewrite the above proof as follows:

$$
\begin{array}{ll}
 & (P \vee Q) \vee (P \vee R) \\
\equiv & \quad \langle \vee \text{ idempotence } \rangle \\
 & P \vee (Q \vee R)
\end{array}
$$

Note that in the above proof we started with the more complex side of the equivalence and sought to transform it into the simpler side. This is a good general technique as stated in the following heuristic (from Gries/Schneider).

Heuristic: To prove $X \equiv Y$, transform the expression with the most structure (either X or Y) into the other.

As another theorem, let's prove that *true* is the zero element for disjunction: $P \vee \mathit{true} \equiv \mathit{true}$.

We focus on the left side of the $\equiv$, which has more structure than the right side. Since it does not match the pattern of any previous law, we try to find a component for which we can substitute. The constant *true* looks promising.

$$
\begin{array}{ll}
 & P \vee \mathit{true} \\
\equiv & \quad \langle \equiv \text{ identity written } \mathit{true} \equiv (P \equiv P) \ \rangle \\
 & P \vee (P \equiv P) \\
\equiv & \quad \langle \vee \text{ distribution over } \equiv \ \rangle \\
 & P \vee P \equiv P \vee P \\
\equiv & \quad \langle \equiv \text{ identity written } (P \vee P \equiv P \vee P) \equiv \mathit{true}\text{, i.e., with } P := P \vee P \ \rangle \\
 & \mathit{true}
\end{array}
$$

The above proof can also be written in reverse order (as can all the proofs we have seen so far). However, that order would not be a good way to present the proof because the first step would involve "pulling a rabbit out of the hat" (i.e., a non-obvious step). We thus adopt the following principle (from Gries/Schneider).

Principle: Structure proofs to minimize the number of rabbits pulled out of the hat. Make each step seem obvious, based on the structure of the expressions and the goal of the manipulation.

Conjunction and the Golden Rule

We define $\wedge$ with a single axiom called the *Golden rule* and give it the same binding power as $\vee$.

Golden rule: $P \wedge Q \equiv P \equiv Q \equiv P \vee Q$

Given the associativity and symmetry of $\equiv$, this is a quite versatile rule.

If we parse the Golden rule as $(P \wedge Q) \equiv (P \equiv Q \equiv P \vee Q)$, we have a definition for $\wedge$. The validity of this law is not obvious, so we justify it by evaluating the expression at some arbitrary point of the state space as shown by the following truth table.

P	Q	$(P \wedge Q)$	$\equiv$	$($	$(P \equiv Q)$	$\equiv$	$(P \vee Q)$	$)$
T	T	T	T		T	T	T	
T	F	F	T		F	F	T	
F	T	F	T		F	F	T	
F	F	F	T		T	F	F	

The following heuristic (from Gries/Schneider) is often useful for proving a property of an operator that is defined in terms of another operator.

Heuristic of Definition Elimination: To prove a theorem concerning an operator $\otimes$ that is defined in terms of another operator $\oplus$, expand the definition of $\otimes$ to arrive at an expression that contains $\oplus$, manipulate the expression using the properties of $\oplus$, and then, if needed, reintroduce $\otimes$ using the definition.

Conjunction is symmetric and associative. These proofs are left to the reader.

Conjunction is also idempotent: $P \wedge P \equiv P$.

$P \wedge P$

$\equiv$ $\langle$ Golden rule written $(P \wedge P) \equiv (P \equiv P \equiv P \vee P)$ $\rangle$

$P \equiv P \equiv P \vee P$

$\equiv$ $\langle$ $\vee$ idempotence $\rangle$

$P \equiv P \equiv P$

$\equiv$ $\langle$ $\equiv$ identity written $(P \equiv P) \equiv \mathit{true}$ $\rangle$

P

The identity element for $\wedge$ is *true*: $P \wedge \mathit{true} \equiv P$.

The proof of the $\wedge$ identity is similar to the proof of idempotence.

What about distribution of $\wedge$ over other operators?

$\wedge$ distributes over itself and $\vee$. Also $\vee$ distributes over $\wedge$. The proofs of these theorems are left to the reader.

$\wedge$ does not distribute over $\equiv$. It does, however, distribute over an even number of $\equiv$'s. We leave the second proof to the reader and address the first a bit later.

Two other useful theorems we can now state are the Absorption laws:

$$P \lor (P \land Q) \equiv P$$
$$P \land (P \lor Q) \equiv P$$

Here we prove the second of these laws.

Although the structure of the left hand side would seem to suggest the use of the distribution law, it turns out not to be directly useful. (Try it!) Thus we fall back on the definition of $\land$, the Golden rule.

$$\begin{array}{ll}
 & P \land (P \lor Q) \\
\equiv & \quad \langle \text{ Golden rule parsed } (P \land Q) \equiv (P \equiv Q \equiv P \lor Q) \ \rangle \\
 & P \equiv P \lor Q \equiv P \lor (P \lor Q) \\
\equiv & \quad \langle \ \lor \text{ associativity and idempotence } \rangle \\
 & P \equiv P \lor Q \equiv P \lor Q \\
\equiv & \quad \langle \ \equiv \text{ identity } (P \equiv P) \equiv \mathit{true} \text{ with } P := P \lor Q \ \rangle \\
 & P
\end{array}$$

The Golden rule has four equivalents. Above we used it by replacing its first equivalent (i.e., the conjunction term) by the remaining three and vice versa. However, because of the associativity and symmetry of $\equiv$, we can use the Golden rule in many ways. We can replace any one equivalent by the other three, any two by the other two, or any three by the other one.

It takes practice to get accustomed to the idea of replacing more than one equivalent. For instance, we can prove something of the form $P \equiv Q \equiv R \equiv S$ by transforming $P \equiv Q$ to $R \equiv S$.

Consider $P \land (Q \equiv R) \equiv P \land Q \equiv P \land R \equiv P$, which shows that $\land$ does not distribute over $\equiv$. We seek to transform the first and last equivalents into the middle two equivalents.

$$\begin{array}{ll}
 & P \land (Q \equiv R) \equiv P \\
\equiv & \quad \langle \text{ Golden rule with } Q := Q \equiv R \text{ (replacing first two equivalents by second two) } \rangle \\
 & (Q \equiv R) \equiv P \lor (Q \equiv R) \\
\equiv & \quad \langle \text{ distribution of } \lor \text{ over } \equiv \ \rangle \\
 & Q \equiv R \equiv P \lor Q \equiv P \lor R \\
\equiv & \quad \langle \ \equiv \text{ symmetry and associativity } \rangle \\
 & P \lor Q \equiv Q \equiv P \lor R \equiv R \\
\equiv & \quad \langle \text{ Golden rule twice (on first two, then last two equivalents) } \rangle \\
 & P \land Q \equiv P \equiv P \land R \equiv P \\
\equiv & \quad \langle \ \equiv \text{ symmetry, associativity, identity } \rangle \\
 & P \land Q \equiv P \land R
\end{array}$$

Thus, it is helpful to remember the following heuristic (from Gries/Schneider).

Heuristic: Exploit the ability to parse theorems like the Golden rule in many different ways.

Implication

We give $\Rightarrow$ a binding power intermediate between $\equiv$ and $\vee$ and define it with a single axiom.

Implication: $P \Rightarrow Q \equiv P \vee Q \equiv Q$

In the table of laws at the end of this document note a number of useful theorems:

$P \Rightarrow P$	(reflexivity)
$P \Rightarrow \mathit{true}$	(true consequent or right zero)
$P \wedge Q \Rightarrow P$	(conjunct simplification)
$P \Rightarrow P \vee Q$	(disjunct addition)

... several more theorems in the table of laws.

Here we prove disjunct addition. Our strategy is to reduce the theorem to *true*.

$$\begin{array}{ll}
 & P \Rightarrow P \vee Q \\
\equiv & \quad \langle \text{ Implication with } Q := P \vee Q \ \rangle \\
 & P \vee P \vee Q \equiv P \vee Q \\
\equiv & \quad \langle \ \vee \text{ idempotence } \rangle \\
 & P \vee Q \equiv P \vee Q \\
\equiv & \quad \langle \ \equiv \text{ identity with } P := P \vee Q \ \rangle \\
 & \mathit{true}
\end{array}$$

Consequence

We give $\Leftarrow$ the same binding power as $\Rightarrow$ and define it with a single axiom.

Consequence: $P \Leftarrow Q \equiv P \vee Q \equiv P$

We give one theorem which connects $\Leftarrow$ with $\Rightarrow$ and, hence, enables us to use the $\Rightarrow$ properties for $\Leftarrow$: $P \Leftarrow Q \equiv Q \Rightarrow P$

$$\begin{array}{ll}
 & P \Leftarrow Q \\
\equiv & \quad \langle \text{ consequence } \rangle \\
 & P \vee Q \equiv P \\
\equiv & \quad \langle \ \vee \text{ symmetry } \rangle \\
 & Q \vee P \equiv P \\
\equiv & \quad \langle \text{ implication } \rangle \\
 & Q \Rightarrow P
\end{array}$$

Negation

We introduce the unary operator $\neg$ to have the highest binding power of the Boolean operators and define it with two axioms—the first to connect it to $\equiv$, the second to $\vee$.

Negation: $\neg(P \equiv Q) \equiv \neg P \equiv Q$

Excluded middle: $P \vee \neg P$

In the table of laws at the end of this document note a number of useful theorems:

$\neg\neg P \equiv P$	(double negation)
$\neg P \wedge \neg Q \equiv \neg(P \vee Q)$	(de Morgan)
$\neg P \vee \neg Q \equiv \neg(P \wedge Q)$	(de Morgan)
$P \vee \neg true \equiv P$	(identity)

... several more theorems in the table of laws.

Above note that $\neg true$ is the identity element for $\vee$. We thus define $\neg true$ to be the predicate $false$ with an axiom.

Definition of $false$: $\neg true \equiv false$

Discrepancy

We give $\not\equiv$ (discrepancy or inequivalence) the same binding power as $\equiv$ and define it with several axioms.

Associativity: $((P \not\equiv Q) \not\equiv R) \equiv (P \not\equiv (Q \not\equiv R))$

Symmetry: $(P \not\equiv Q) \equiv (Q \not\equiv P)$

Association with $\equiv$: $P \equiv (Q \not\equiv R) \equiv (P \equiv Q) \not\equiv R$

Distribution over $\equiv$: $P \not\equiv Q \equiv \neg(P \equiv Q)$

In the table of laws at the end of this document note two useful theorems:

$P \not\equiv false \equiv P$	(identity)
$P \wedge (Q \not\equiv R) \equiv P \wedge Q \not\equiv P \wedge R$	(distribution of $\wedge$ over $\not\equiv$)

Leibniz as an Axiom

We have introduced the Leibniz rule as an inference rule in the logic. Sometimes it is convenient to use Leibniz as an axiom.

Leibniz: $e = f \Rightarrow E^z_e = E^z_f$

Remember that $\equiv$ is the same *semantic* concept as $=$ when predicates are being considered.

Proof Methods for Implications

Prove the *monotonicity of* $\vee$: $(P \Rightarrow Q) \Rightarrow (P \vee R \Rightarrow Q \vee R)$

We use four different methods to prove this theorem, comparing the methods.

1. Method: show the theorem equivales ($\equiv$) *true*.

   Our strategy is to expand using the $\Rightarrow$ to $\neg$ connection and then simplify using complement and excluded middle.

$$
\begin{array}{ll}
 & (P \Rightarrow Q) \Rightarrow (P \vee R \Rightarrow Q \vee R) \\
\equiv & \qquad \langle \Rightarrow \neg \text{ connection } \rangle \\
 & \neg(P \Rightarrow Q) \vee (P \vee R \Rightarrow Q \vee R) \\
\equiv & \qquad \langle \Rightarrow \neg \text{ connection twice } \rangle \\
 & \neg(\neg P \vee Q) \vee (\neg(P \vee R) \vee (Q \vee R)) \\
\equiv & \qquad \langle \text{ de Morgan twice, double negation } \rangle \\
 & (P \wedge \neg Q) \vee ((\neg P \wedge \neg R) \vee (Q \vee R)) \\
\equiv & \qquad \langle \text{ complement written } (\neg P \wedge \neg R) \vee R \equiv \neg P \vee R \ \rangle \\
 & (P \wedge \neg Q) \vee (\neg P \vee R \vee Q) \\
\equiv & \qquad \langle \text{ complement written } (P \wedge \neg Q) \vee Q \equiv P \vee Q \ \rangle \\
 & P \vee Q \vee \neg P \vee R \\
\equiv & \qquad \langle \text{ excluded middle } \rangle \\
 & \mathit{true} \vee Q \vee R \\
\equiv & \qquad \langle \vee \text{ zero element } \rangle \\
 & \mathit{true}
\end{array}
$$

   Observations:

   - The proof is long. That is, it contains many lines.
   - The proof is wide. That is, most lines are relatively long. Thus each line is relatively difficult to understand.

2. Method: show the left-hand-side (LHS) of the central "⇒" implies (⇒) the right-hand-side (RHS).

   This is a new technique. The steps can be justified by one of the equivalence transformations (using the Leibniz inference rule) or by one of the implication rules like disjunct addition or conjunct simplification.

$$
\begin{array}{ll}
 & P \Rightarrow Q \\
\equiv & \qquad \langle \Rightarrow \neg \text{ connection } \rangle \\
 & \neg P \vee Q \\
\Rightarrow & \qquad \langle \text{ Notes 1, 2, \& 3, disjunct addition } \rangle \\
 & (\neg P \vee Q) \vee R \\
\equiv & \qquad \langle \text{ complement written } \neg P \vee R \equiv (\neg P \wedge \neg R) \vee R \ \rangle \\
 & (\neg P \wedge \neg R) \vee Q \vee R \\
\equiv & \qquad \langle \text{ de Morgan } \rangle \\
 & \neg(P \vee R) \vee (Q \vee R) \\
\equiv & \qquad \langle \Rightarrow \neg \text{ connection } \rangle \\
 & P \vee R \Rightarrow Q \vee R
\end{array}
$$

   Note 1: To use an implication theorem to justify a step, the preceding predicate must satisfy the *entire pattern* of the LHS of the implication in the theorem, not just a subexpression. (Remember that replacement of subexpressions by an equivalent is justified by the Leibniz inference rule, whose premise is an equality and conclusion is an equality involving textual substitution.)

   Note 2: We motivate this step by noting that the goal expression has a predicate R that must be added to the expression somehow.

   Note 3: In an implication $P \Rightarrow Q$, the predicate P is said to be *stronger* than predicate Q and Q is said to be *weaker* than P. Thus a ⇒ step justified by disjunct addition, conjunct simplification, or similar rules is called a *weakening* step. The reverse transformation (from the line below to the line above) is called a *strengthening* step.

   Note 4: The formal bases for this proof method are the transitivity theorem for ⇒ and the mutual implication theorem (i.e., $(P \equiv Q) \equiv (P \Rightarrow Q) \wedge (Q \Rightarrow P)$).

   Observations:

   - (Advantage) Proof is shorter.
   - (Advantage) Proof is narrower. Each line tends to be easier to understand.
   - (Disadvantage) A few of the transformations are difficult to motivate. In particular, the "disjunct addition" step is a big rabbit pulled out of a hat.

Principle: Short, narrow proofs are usually more elegant and easier to understand than long, wide proofs.

(But, of course, terseness can be carried too far. Be sure to give a sufficient number of steps with appropriate justifications to make the proof easy to read and understand.)

Heuristic: The first proof we discover is probably not the best. Look for a better one.

3. Method: show the RHS follows from ($\Leftarrow$) the LHS

   This proof method is like the previous one except that we allow transformations justified by $\Leftarrow$ instead of $\Rightarrow$. We start with the RHS.

$$
\begin{array}{ll}
 & P \lor R \Rightarrow Q \lor R \\
\equiv & \qquad \langle \text{ } \Rightarrow \neg \text{ connection } \rangle \\
 & \neg(P \lor R) \lor (Q \lor R) \\
\equiv & \qquad \langle \text{ de Morgan } \rangle \\
 & (\neg P \land \neg R) \lor Q \lor R \\
\equiv & \qquad \langle \text{ complement written } \neg P \lor R \equiv (\neg P \land \neg R) \lor R \text{ } \rangle \\
 & (\neg P \lor Q) \lor R \\
\Leftarrow & \qquad \langle \text{ strengthening (i.e., disjunct addition used right to left) } \rangle \\
 & \neg P \lor Q \\
\equiv & \qquad \langle \text{ } \Rightarrow \neg \text{ connection } \rangle \\
 & P \Rightarrow Q
\end{array}
$$

   Observations:

   - (Advantage) Same length and width as the weakening proof (method 2 using $\Rightarrow$ steps).
   - (Advantage) The strengthening ($\Leftarrow$ steps) are better motivated by the structure of the expression than the weakening steps are in the previous proof. A rabbit is not pulled from the hat. This proof is easier to discover (given enough practice with consequent proofs) and easier to understand in a top-to-bottom reading.

4. Method: assume LHS, show RHS equivales *true* using assumption.

   We can also prove a theorem $P \Rightarrow Q$ by "assuming the antecedent" (i.e., the LHS) and then proving that the consequent (i.e., the RHS) equivales *true*, using the antecedent in the proof as if it were an axiom.

   Restriction: In applying this method, the variables appearing in the assumptions are taken to be constants, so Substitution cannot be used to replace these variables.

   A proof of Q which uses assumptions $P_0, \ldots\ P_{n-1}$ can be mechanically transformed into a proof of $P_0 \wedge \ldots\ \wedge P_{n-1} \Rightarrow Q$. This translation is the formal basis for the proof method.

   Using this method, we write the proof in the following format.

   Assume $P \Rightarrow Q$ (which is equivalent to $(P \Rightarrow Q) \equiv \mathit{true}$)

$$
\begin{array}{ll}
 & P \vee R \Rightarrow Q \vee R \\
\equiv & \quad \langle \Rightarrow \neg \text{ connection } \rangle \\
 & \neg(P \vee R) \vee (Q \vee R) \\
\equiv & \quad \langle \text{ de Morgan } \rangle \\
 & (\neg P \wedge \neg R) \vee Q \vee R \\
\equiv & \quad \langle \text{ complement written } \neg P \vee R \equiv (\neg P \wedge \neg R) \vee R \ \rangle \\
 & (\neg P \vee Q) \vee R \\
\equiv & \quad \langle \Rightarrow \neg \text{ connection } \rangle \\
 & (P \Rightarrow Q) \vee R \\
\equiv & \quad \langle \text{ Assumption } \rangle \\
 & \mathit{true} \vee R \\
\equiv & \quad \langle \vee \text{ zero } \rangle \\
 & \mathit{true}
\end{array}
$$

   Note: Always state the assumptions explicitly at the beginning of the proof and be sure to give an appropriate hint in steps that use the assumptions.

   Observations:

   - (Advantage) This proof is approximately the same length as the weakening and strengthing proofs. (In this case it is one step longer, but we will often combine the use of the assumption with another transformation such as the "∨ zero" step above.)
   - (Advantage) If the antecedent (LHS) is composed of several conjuncts, this proof method allows the conjuncts to be introduced into the proof separately, as they are needed in the manipulation.

Other Proof Methods

Proof by case analysis:

To prove R, find two cases P and Q such that $P \vee Q$ holds and that $P \Rightarrow R$ and $Q \Rightarrow R$ hold. This expands to more than two cases as appropriate.

In case analysis proofs it is very important that we explicitly identify the cases and make sure that all cases have been covered. Thus the following format should be used for writing the proof of R.

Prove: R
  By cases: P, Q
    (give proof of $P \vee Q$ unless obvious)
  Case P**:** (give proof of $P \Rightarrow R$)
  Case Q**:** (give proof of $Q \Rightarrow R$)

Proof by mutual implication (ping-pong method):

Another way to prove $P \equiv Q$ is to prove $P \Rightarrow Q$ and $Q \Rightarrow P$.

The formal basis for this proof method is the mutual implication theorem,
$P \equiv Q \equiv (P \Rightarrow Q) \wedge (Q \Rightarrow P)$.

Historically, this has been a common method of proof in mathematics. However, we consider it an *inelegant* method of proof. In most instances it can and *should* be replaced by a straightforward equational proof of equivalence using one of the previous proof methods.

This proof method is necessary in situations where the $P \Rightarrow Q$ and $Q \Rightarrow P$ proofs must rely on different properties.

Proof by contradiction:

To prove P, assume $\neg P$ and derive a contradiction, that is, show that $\neg P$ implies *false* (or implies something that is equivalent to *false*).

The formal basis for this method is the theorem $\neg P \Rightarrow false \equiv P$. The proof of this theorem is left to the reader.

Historically, this has also been a common method of proof in mathematics. But it tends to be overused. Often a proof of contradiction can be more simply written using one of the direct methods of proof.

One situation in which this proof method is sensible is in proving the existence (or nonexistence) of an entity having certain characteristics. We assume that one does not exist (or does exist) and show that the assumption leads to a contradiction.

However, in programming, we are usually not satisfied with just knowing about the existence of some kind of entity. If the entity exists, we usually want an algorithm to construct it. Direct, or constructive, methods of reasoning are usually more helpful toward this end.

Proof by contrapositive:

To prove $P \Rightarrow Q$, prove $\neg Q \Rightarrow \neg P$.

The formal basis for this proof method is the contraposition theorem, $P \Rightarrow Q \equiv \neg Q \Rightarrow \neg P$.

Quantified Expressions

Let's step back from predicate calculus and take a look at *quantified expressions* in a more general context. Examples of quantified expressions from mathematics and logic are summations (written with Σ), products (Π), universal quantifications ($\forall$), and existential quantifications ($\exists$).

We write the quantified version of an operator *op* as follows:

$$(\mathbf{OP}\ i : R : T)$$

- We call this a quantified expression or *quantification* for short.
- The operator *op* must be a symmetric (i.e., commutative) and associative binary operator.
- **OP** is called the *quantifier*. (We show it here capitalized and in bold face. In handwritten text use underlining instead of bold.)
- The quantified expression has the same result type as the operator *op*. That is, if *op* is arithmetic, then the quantified expression is arithmetic; if *op* is Boolean (i.e., a predicate operator), then the result is Boolean.
- i is a list of one or more *dummy variables* that are given values by the quantifier within the scope of the quantification—the beginning and ending parentheses of the quantified expression. We speak of the *occurrences* of the dummy variables appearing within the quantified expression as being *bound* with respect to the quantified expression or to any larger expression in which the quantified expression appears.

  We say that an *occurrence* of a variable within some expression is *free* if that occurrence is not bound by a quantifier in that expression .

  Within a quantified expression, a dummy variable having the same name as a free variable (or a dummy bound in an outer scope) hides the name of the free variable.
- R is called the *range* of the quantification. It is a predicate and is normally a function of the dummy variables. If the range is omitted, we take it to be the predicate *true*.

  When we say that the range is *empty*, we mean that the range is the predicate *false* (or another predicate that equivales *false* for all values of the dummies).

  When we say that the range is *finite*, we mean that it equivales *true* for a finite number of values of the quantified variables.
- T is the *term* (sometimes called the *body*) of the quantification. The term must have the same type as the operands of the operator *op*. It is usually a function of the dummy variables.
- Speaking operationally, the quantification generates a list containing the values of term expression T for all values of the dummies i that satisfy the range predicate R and then inserts the associative and symmetric binary operator *op* between adjacent elements of the list to compute the value of the quantification.

Since the same variable name can be used as both a dummy and as a free variable within some expression, we must refine the definition of the textual substitution operation and the Leibniz inference rule to take this into account.

Textual substitution: If $\neg$*occurs*(*'y'*, *'x,F'*),
$(\mathbf{OP}\ y : R : T)(x := F) = (\mathbf{OP}\ y : R(x := F) : T(x := F))$.

Here we use a predicate of the form *occurs*(*'v'*, *'E'*) to mean that at least one of the variables in the list v occurs free (i.e., unquantified) in the list of expressions E.

Thus the caveat $\neg$*occurs*(*'y'*, *'x,F'*) means that no dummy variable in list y appears free in either expression x or in expression F.

However, if a dummy in list y does appear free in expression *'x,F'*, all is not lost. We can apply the Dummy Renaming Theorem below to rename that dummy. All occurrences of the troublesome dummy can simply be replaced by a *fresh* variable (i.e., a variable that does not appear in any of the expressions under consideration).

When possible, it is better to avoid use of the same variable names for two different purposes.

Consider the expression E where $E \equiv n \neq 7 \wedge (\forall n : n < 10 : m > n + 2)$.

- The occurrence of variable n in subexpression $n \neq 7$ is free with respect to E.
- The occurrences of n in subexpression $(\forall n : n < 10 : m > n + 2)$ are bound with respect to E —and to $(\forall n : n < 10 : m > n + 2)$.
- The occurrence of m is free with respect to E.
- The occurrence of n in subexpression $m > n + 2$ is free with respect to subexpression $m > n + 2$ even though it is bound with respect to the entire expression E.
- In $E(m := z + 1)$, bound variable n does not occur free in m or $z + 1$.

  Thus, $E(m := z + 1) \equiv n \neq 7 \wedge (\forall n : n < 10 : (z + 1) > n + 2)$.
- In $E(n := z + 1)$, bound variable n does occur free in n or $z + 1$. This requires that we first change the bound occurrences of n to be occurrences of a fresh variable, say i.

  Thus, $E(n := z + 1) \equiv (z + 1) \neq 7 \wedge (\forall i : i < 10 : m > i + 2)$.

  If we are careful to treat the free and bound occurrences of n differently, it is not actually necessary to introduce a new variable.

  Thus, $E(n := z + 1) \equiv (z + 1) \neq 7 \wedge (\forall n : n < 10 : m > n + 2)$.
- In $E(m := n + 1)$ bound variable n does occur free in m or $n + 1$. This also requires that we rename the bound variable n,

  Thus, $E(m := n + 1) \equiv n \neq 7 \wedge (\forall i : i < 10 : (n + 1) > i + 2)$.
- $E(m, n := n + 1, z + 1) \equiv z + 1 \neq 7 \wedge (\forall i : i < 10 : (n + 1) > i + 2)$.
- $(E(m := n + 1))(n := z + 1) \equiv z + 1 \neq 7 \wedge (\forall i : i < 10 : ((z + 1) + 1) > i + 2)$.

Because of the caveat $\neg$*occurs('y','x,F')* on textual substitution in quantified expressions (and the resulting requirement for renaming bound variables), the previous statement of the Leibniz inference rule is not sufficient for quantified expressions.

To see the problem, let us assume that f and g are Boolean functions on one variable and that we wish to prove $(\forall x : f.x : g.(x+x)) = (\forall x : f.x : g.(2*x))$. To do so, we would need to use the following instance of the Leibniz inference rule.

$$\frac{x + x = 2 * x}{(\forall x : f.x : g.z)(z := x + x) \;=\; (\forall x : f.x : g.z)(z := 2 * x)}$$

But because of the definition of textual substitution over quantified expressions, what we must prove then is $(\forall i : f.i : g.(x+x)) = (\forall j : f.j : g.(2*x))$, which is not at all what we want.

Therefore, we add two new Leibniz axioms to handle quantifications appropriately.

Leibniz:
$$\frac{P = Q}{(\mathbf{OP}\; i : R(z := P) : T) \;=\; (\mathbf{OP}\; i : R(z := Q) : T)}$$

$$\frac{R \Rightarrow P = Q}{(\mathbf{OP}\; i : R : T(z := P)) \;=\; (\mathbf{OP}\; i : R : T(z := Q))}$$

Thus for proving $(\forall x : f.x : g.(x+x)) = (\forall x : f.x : g.(2*x))$ as discussed above, we can use an instance of the second rule. (If $x + x = 2 * x$, then $f.x \Rightarrow x + x = 2 * x$.)

We can give general axioms that hold for all quantifications of associative and symmetric binary operations.

Term rule: $(\mathbf{OP}\; i : R : P \; op \; Q) = (\mathbf{OP}\; i : R : P) \; op \; (\mathbf{OP}\; i : R : Q)$.

We give the hints *splitting the term* or *joining the term* when we use this axiom in arithmetic or logical calculations.

Range rule: If (1) *op* is idempotent or (2) $P \wedge Q \equiv false$ (for all i) and either P and Q are finite or *op* is a Boolean operator, then $(\mathbf{OP}\; i : P \vee Q : T) = (\mathbf{OP}\; i : P : T) \; op \; (\mathbf{OP}\; i : Q : T)$. We use the hints *splitting the range* or *joining the range* when applying this rule.

Range rule (generalized): If *op* is a Boolean operator or if P and Q are finite, $(\mathbf{OP}\; i : P \vee Q : T) \; op \; (\mathbf{OP}\; i : P \wedge Q : T) = (\mathbf{OP}\; i : P : T) \; op \; (\mathbf{OP}\; i : Q : T)$.

Distributivity: If $\neg$*occurs('i','Z')*, $\oplus$ distributes over *op*, and the range R is nonempty, $(\mathbf{OP}\; i : R : Z \oplus T) = Z \oplus (\mathbf{OP}\; i : R : T)$.

Empty range: If u is the identity element for *op*, $(\mathbf{OP}\; i : false : T) = u$.

Constant term: If $\neg$*occurs('i','Z')*, *op* is idempotent, and the range R is nonempty, $(\mathbf{OP}\; i : R : Z) = Z$.

Dummy transformation rule: If $\neg$*occurs('j','R,T')* and function f is invertible, $(\mathbf{OP}\; i : R : T) = (\mathbf{OP}\; j : R(i := f.j) : T(i := f.j))$.

Nesting rule: If $\neg occurs(\text{'}j\text{'}, \text{'}R\text{'})$, $(\mathbf{OP}\ i : R : (\mathbf{OP}\ j : Q : T)) = (\mathbf{OP}\ i, j : R \wedge Q : T)$.

One-point rule: If $\neg occurs(\text{'}i\text{'}, \text{'}Z\text{'})$, $(\mathbf{OP}\ i : i = Z : T) = T(i := Z)$.

Interchange: If $\neg occurs(\text{'}j\text{'}, \text{'}P\text{'})$ and $\neg occurs(\text{'}i\text{'}, \text{'}Q\text{'})$,
$(\mathbf{OP}\ i : P : (\mathbf{OP}\ j : Q : T)) \ = \ (\mathbf{OP}\ j : Q : (\mathbf{OP}\ i : P : T))$.

Given the above general axioms, we can prove various theorems about quantification.

Dummy renaming: If $\neg occurs(\text{'}j\text{'}, \text{'}R, T\text{'})$, $(\mathbf{OP}\ i : R : T) = (\mathbf{OP}\ j : R(i := j) : T(i := j))$.

Generalized distributivity:
If (1) $\oplus$ distributes over *op* and (2) either the range is nonempty or the identity element of *op* is the zero element of $\oplus$, $(\mathbf{OP}\ i : R : Z \oplus T) = Z \oplus (\mathbf{OP}\ j : R : T)$.

We have shown the quantifier associated with an operator by capitalizing and emboldening (or underlining) the operator. Historically, however, the quantified versions of the following operators have been given special symbols. We use those symbols.

op	**OP**
$\wedge$	$\forall$
$\vee$	$\exists$
$+$	Σ
$*$	Π

Universal Quantification

Universal quantification is the name given to the quantified version of $\wedge$. Conjunction is associative, symmetric, and idempotent; it has identity element *true* and zero element *false*.

Universal quantifications are written in the form $(\forall\, i : R : T)$ where both R and T are predicates. We read this expression as "for all i such that R holds, T holds".

Thus we have the following instances of the general quantification axioms and theorems given in the previous section:

- Term splitting/joining
- Range splitting/joining
- Distribution of $\wedge$ over $\forall$ with nonempty range
- Empty range
- Constant term
- Dummy transformation
- Nesting
- One point
- Interchange
- Dummy renaming
- Distribution of $\vee$ over $\forall$

We include another axiom, trading, that allows us to move expressions between the range and the term. Laws of this type are only possible for quantified forms of the Boolean operators, in which the range and the term have the same type.

Trading: $(\forall\, i : R : T) \equiv (\forall\, i :: \neg R \vee T)$

There are a few other useful theorems given in the tables at the end of this document.

As an example proof, consider the following theorem: $(R \Rightarrow T) \Rightarrow (\forall\, i : R : T)$

Assume $R \Rightarrow T$ (which is equivalent to $R \Rightarrow T \equiv true$)

$$
\begin{array}{ll}
 & (\forall\, i : R : T) \\
\equiv & \quad \langle \text{ trading } \rangle \\
 & (\forall\, i :: \neg R \vee T) \\
\equiv & \quad \langle \Rightarrow \neg \text{ connection } \rangle \\
 & (\forall\, i :: R \Rightarrow T) \\
\equiv & \quad \langle \text{ assumption } \rangle \\
 & (\forall\, i :: true) \\
\equiv & \quad \langle \text{ constant term } \rangle \\
 & true
\end{array}
$$

As another example, consider the following theorem:

$$(\forall\, x : x = 2 * y \wedge y > 0 : x \ \mathbf{mod}\ 4 = 0) \equiv y \ \mathbf{mod}\ 2 = 0 \vee y \leq 0$$

$$\begin{array}{ll}
 & (\forall\, x : x = 2 * y \wedge y > 0 : x \ \mathbf{mod}\ 4 = 0) \\
\equiv & \quad \langle \text{ trading, de Morgan } \rangle \\
 & (\forall\, x :: \neg(x = 2 * y) \vee \neg(y > 0) \vee x \ \mathbf{mod}\ 4 = 0) \\
\equiv & \quad \langle \text{ trading, arithmetic } \rangle \\
 & (\forall\, x : x = 2 * y : y \leq 0 \vee x \ \mathbf{mod}\ 4 = 0) \\
\equiv & \quad \langle\ x \text{ does not occur free in } y \leq 0 \text{, distribution of } \vee \text{ over } \forall\ \rangle \\
 & y \leq 0 \vee (\forall\, x : x = 2 * y : x \ \mathbf{mod}\ 4 = 0) \\
\equiv & \quad \langle \text{ one point } \rangle \\
 & y \leq 0 \vee (x \ \mathbf{mod}\ 4 = 0)(x := 2 * y) \\
\equiv & \quad \langle \text{ substitution } \rangle \\
 & y \leq 0 \vee 2 * y \ \mathbf{mod}\ 4 = 0 \\
\equiv & \quad \langle \text{ arithmetic } \rangle \\
 & y \ \mathbf{mod}\ 2 = 0 \vee y \leq 0
\end{array}$$

Existential Quantification

Existential quantification is the name given to the quantified version of $\vee$. Disjunction is associative, symmetric, and idempotent; it has identity element *false* and zero element *true*.

Existential quantifications are written in the form $(\exists\, i : R : T)$ where both R and T are predicates. We read this expression as "there exists an i in the range R such that T holds".

Thus we have the following instances of the general quantification axioms and theorems given in the earlier section:

- Term splitting/joining
- Range splitting/joining
- Distribution of $\vee$ over $\exists$ with nonempty range
- Empty range
- Constant term
- Dummy transformation
- Nesting
- One point
- Interchange
- Dummy renaming
- Distribution of $\wedge$ over $\exists$

We include another useful axiom, generalized de Mogran.

de Morgan: $(\exists\, i : R : T) \;\equiv\; \neg(\forall\, i : R : \neg T)$

The dual of the above axiom is a theorem: $(\forall\, i : R : T) \;\equiv\; \neg(\exists\, i : R : \neg T)$.

There are a few other useful theorems given in the tables at the end of this document.

Arithmetic Quantifications

Summation is the name given to the quantified version of $+$. We use Σ as the quantification operator. A summation quantification is only defined for a finite range.

The addition $(+)$ operation is associative and symmetric but not idempotent. It has identity element 0 but no zero element.

The usual quantification axioms and theorems apply to summation.

Product is the name given to the quantified version of $*$. We use Π as the quantification operator. A product quantification is only defined for a finite range.

The multiplication $(*)$ operation is associative and symmetric but not idempotent. It has identity element 1 and zero element 0.

The usual quantification axioms and theorems apply to product.

Let binary operations **max** and **min** be defined on integers such that

$$\begin{array}{lcl} x \;\mathbf{max}\; y = z & \equiv & (x = z \vee y = z) \wedge x \leq z \wedge y \leq z \\ x \;\mathbf{min}\; y = z & \equiv & (x = z \vee y = z) \wedge x \geq z \wedge y \geq z \end{array}$$

Note that the operations **max** and **min** are both associative, symmetric, and idempotent. We assign the special value $-\infty$ as the identity element for **max** and the zero element for **min**. Similarly, we assign the special value $+\infty$ as the zero element for **max** and the identity element for **min**. For any integer n, we let $-\infty < n$ and $n < +\infty$. We leave the arithmetic operations (e.g., addition, subtraction, multiplication, division, mod, etc.) undefined for $-\infty$ and $+\infty$.

Furthermore, note that **max** and **min** distribute over each other:

$$\begin{array}{lcl} x \;\mathbf{min}\; (y \;\mathbf{max}\; z) & = & (x \;\mathbf{min}\; y) \;\mathbf{max}\; (x \;\mathbf{min}\; z) \\ x \;\mathbf{max}\; (y \;\mathbf{min}\; z) & = & (x \;\mathbf{max}\; y) \;\mathbf{min}\; (x \;\mathbf{max}\; z) \end{array}$$

Maximization is the name given to the quantified version of **max**. We use **MAX** as the quantification operator.

Minimization is the name given to the quantified version of **min**. We use **MIN** as the quantification operator.

The usual quantification axioms and theorems apply to maximization and minimization. In addition, we add the following axiom to relate maximization and minimization.

MAX/MIN relation: $(\mathbf{MIN}\, i : R : -E) = -(\mathbf{MAX}\, i : R : E)$

Let operation # be defined such that $\#.true = 1$ and $\#.false = 0$. This unary operator allows us to convert Booleans to integers.

Numerical quantification is the quantified version of the unary operator #. We also use # as the quantification operator. (Some writers use **N** as the quantification operator.) A numerical quantification is only defined for a finite range.

Numerical quantification is not defined in terms of its underlying operator as the other quantifications are. Instead it is defined in terms of summation with the following axiom:

Numerical quantification: $(\#\, i : R : T) = (\Sigma\, i : R \wedge T : 1)$
where R and T are both Boolean expressions.

Speaking operationally, numerical quantification counts the number of values of the dummy variables for which both the range predicate R and the term predicate T hold.

The one-point rule for numerical quantification can be defined as follows:

If $\neg occurs(\text{'}i\text{'}, \text{'}Z\text{'})$, $(\#\, i : i = Z : T) \;=\; \#.(T(i := Z))$.

We also introduce the following axiom relating the quantified version of # to the unquantified version of the operator.

Numerical quantification: $(\#\, i : R : T) = (\Sigma\, i : R : \#.T)$

FORMALIZING ENGLISH

Introduction

Note: This section is based, in part, on sections 2.4, 5.1, and 9.3 of the Gries and Schneider textbook.

Our logic is more than an entertaining symbol manipulation game. It provides us a means for reasoning about English statements. We can formalize English statements as logical expressions, manipulate the formalizations using the methods we have discussed, and then interpret the resulting logical expression.

This has at least two uses.

1. We can formalize an English statement as a logical expression and then prove that the logical expression is a theorem.

   If the expression is not a theorem, then the English statement is unsound. In this case, our attempt at proving the expression may lead to a counterexample—a state in which the expression is FALSE.

2. We can also formalize the statement of some problem and use logical manipulations to solve the problem.

In programming, we can state the requirements for a program as logical expressions in terms of the program's constants and variables. This formalization is usually called the program's *specification*.

The first use of logic above would then be what is often called *program verification*. That is, for some program P and specification S, we must prove that P *satisfies* S is a theorem (where *satisfies* has been given a precise meaning in our logic).

The second use of logic above would then be what is often called *program derivation*. In program derivation we use the formal specification to find an appropriate program that satisfies the specification.

Before we discuss programs further, let's look at the process of formalizing English statements and manipulating the formalizations.

Translating English into Logic

An expression in our logic represents a Boolean function. The free variables occurring in the expression are the arguments of the function. If the free variables are all of type Boolean, then we can call the logical expression a *proposition*.

We can associate a proposition with an English statement that can be interpreted as either being TRUE or being FALSE.

For example, consider the following English proposition:

> Mary has two sons and Keith has two daughters.

We can formalize this proposition in our logic as follows. We introduce two logical expressions, the Boolean variables m and k, to represent the two subpropositions of the above proposition.

m : Mary has two sons
k : Keith has two daughters

Then we can represent the entire statement by the expression

$$m \wedge k$$

where we have replaced the English connective "and" by the Boolean operator $\wedge$.

The process of translating a proposition p into a logical expression can be summarized as follows:

1. Introduce Boolean variables to denote subpropositions.

2. Replace these subpropositions by the corresponding Boolean variables.

3. Translate the resulting expression into a Boolean expression, using the "obvious" translations of the English words into operators:

"and"	becomes	$\wedge$
"or"	becomes	$\vee$
"not" "it is not the case that"	becomes	$\neg$
"if p then q" "p is sufficient for q"	becomes	$p \Rightarrow q$
"p only if q" "p when q" "p is necessary for q"	becomes	$q \Rightarrow p$ (or $p \Leftarrow q$)
"p if and only if q" "p is necessary and sufficient for q"	becomes	$p \equiv q$

Of course, not all translations are obvious; nor are all English statements unambiguous.

For example, the English word "or" is used in two different ways, in an *inclusive* sense and in an *exclusive* sense.

Consider the sentence "Professor Cunningham will give his class a difficult exam or he will give a long exam". This is the inclusive sense of "p or q" since the exam could be both difficult and long. (And Professor Cunningham's exams often are both!) This sense can be formalized as $p \vee q$.

Consider the sentence "I'll flunk the predicate logic exam or I'll pass it". This is the exclusive sense of "p or q" since only one of the options can occur at once. This sense can be formalized as $p \not\equiv q$ or, equivalently, as $p \equiv \neg q$.

Sometimes implications are hidden. Consider the following statements:

> If you don't pass the exam, you'll flunk the class.
> Pass the exam or you'll flunk the class

The first has the form $\neg p \Rightarrow q$, the second has the form $p \vee q$. Of course, these two expressions are equivalent in our logic.

Or consider the statement "a person residing in Oxford resides in Lafayette County." This can be interpreted the same as "If a person resides in Oxford, then the person resides in Lafayette County." The restructuring of the sentence makes the two subpropositions clear.

Equivalence (and equality) is handled awkwardly by the English language. As a result, equivalences are often hidden in what looks like an implication.

For example, we might see the sentence "If two sides of a triangle are equal, the triangle is isosceles" when the intent is to define what is meant by an isosceles triangle. What seems to have the form $p \Rightarrow q$ really is meant to be $p \equiv q$.

To state the above precisely in English we might say one of the following, none of which seems very natural:

> Two sides of a triangle are equal if and only if the triangle is isosceles.
> Two sides of a triangle are equal exactly when the triangle is isosceles.
> "Two sides of a triangle are equal" is the same as "the triangle is isosceles."

The logic gives us a concise and precise way to state the proposition.

Portia's Suitor's Dilemma

Consider the following puzzle, known as Portia's Suitor's Dilemma. This is a simplification of a situation in Shakespeare's play *The Merchant of Venice*. (The formulation here is adapted from the Gries and Schneider textbook.)

Portia has a suitor who wishes to marry her. She decides to test his worthiness by asking him to solve a puzzle. Portia has a gold casket and a silver casket. On the caskets she has the following inscriptions:

> Gold: The portrait is not in here.
> Silver: Exactly one of these inscriptions is true.

Portia explains to her suitor that each inscription may be true or false, but that she has placed her portrait in one of the caskets in a manner that is consistent with the truth or falsity of the inscriptions. If he can choose the casket with her portrait, she will marry him.

Which casket contains the portrait?

To solve the puzzle, we first formalize it by introducing Boolean variables to represent the various propositions.

gc : The portrait is in the gold casket.
sc : The portrait is in the silver casket.
g : The portrait is not in the gold casket. —the inscription on the gold casket
s : Exactly one of g and s is *true*. —the inscription on the silver casket

We have three facts that we must formalize in terms of these variables.

1. There is only one portrait. Thus it can only be in one place.

   $F_0 : gc \equiv \neg sc$

2. The inscription on the gold casket is the negation of gc.

   $F_1 : g \equiv \neg gc$

3. The inscription on the silver casket says that exactly one of the inscriptions is true. Thus the truth of the silver inscription equivales the expression $s \equiv \neg g$.

   $F_2 : s \equiv (s \equiv \neg g)$

The facts F_0, F_1, and F_2 formalize Portia's puzzle. We take these as axioms of the logic and calculate to see whether we can derive either sc or gc.

F_2 has the most structure. Thus we start with it.

$$\begin{array}{ll} & s \equiv s \equiv \neg g \\ \equiv & \quad \langle\ \equiv \text{ identity written } (s \equiv s) \equiv \mathit{true}\ \rangle \\ & \neg g \\ \equiv & \quad \langle\ F_1 \text{ rewritten } \neg g \equiv gc\ \rangle \\ & gc \end{array}$$

Hence, from F_1 and F_2, we can conclude gc. The portrait is in the gold casket. (Note that F_0 is not needed for this calculation.)

Of course, the above argument is based upon the assumption that the three facts hold—that they can be axioms. Hence, the three facts must be consistent with one another. That is, it must be possible for all three to hold simultaneously when $gc \equiv \mathit{true}$. It is sufficient to show that some assignment of values to the variables make them all hold. The assignment $sc = false$, $g = false$, and $s = false$ (or $s = true$) will do what is desired.

Suppose we change the puzzle so that the inscription on the silver casket (i.e., s) is formalized as follows.

> s': This inscription is false.

Thus F_2 must be replaced by F_2'.

> $F_2' : s \equiv \neg s$

But F_2' equivales *false* and is, hence, an absurd axiom (i.e., truth).

With *false* as a axiom, we can conclude anything (since *false* $\Rightarrow P$). Thus with F_2' as a "fact", we conclude that the puzzle can have no solution.

Translating Quantifications

Above we formalized English statements in terms of propositions, that is, Boolean-valued expressions whose only free variables are themselves of type Boolean. Also we did not consider quantifications above.

For example, earlier we considered the proposition:

> Mary has two sons and Keith has two daughters.

The statement "Mary has two sons" is treated above as an indivisible primitive statement that we interpret as either being true or false. However, we might want to reason about the constituent elements of such statements. For example, we might want to formalize the "has two" relationship in a more general way, such as defining predicate *has_two* as follows:

> $\mathit{has\_two}.x.y$: x has two y

Then the above proposition could be written as:

> $\mathit{has\_two}.\mathrm{Mary.sons} \;\wedge\; \mathit{has\_two}.\mathrm{Keith.daughters}$

We also want to use quantifications and treat the various fields of the quantification as predicates. For example, if we wanted to say "all cats have two eyes", we can formalize that statement as

> $(\forall\, x : \mathit{cat}.x : \mathit{has\_two}.x.\mathrm{eyes}\ )$

where $\mathit{cat}.x$ is *true* if and only if x is a cat.

As another example, consider the mathematical statement "some natural number between 80 and n is a multiple of x".

We can formalize this statement as

> $(\exists\, i : 80 \leq i \leq n : \mathit{mult}.i.x)$

where $\mathit{mult}.n.x$ is formalized as

> $(\exists\, m :: n = m * x)$.

Formalizing English can help in two ways.

- It can expose ambiguities and force precision.
- It allows the predicate logic to be used in reasoning about the objects under consideration.

In a previous subsection we discussed the correspondence between various English phrases and the unary and binary operators of the logic. Now we can discuss the correspondence between English phrases and the quantification operators of the logic.

We can substitute the operator $\forall$, read "for all", for phrases such as

> *for all*, *for each*, *for arbitrary*, *every*, and *any*.

Not all universal quantifications are signaled by an explicit phrase however. Sometimes the presence of the indefinite articles "a" and "an" is a clue that a universal quantification is meant implicitly. Consider the following statements:

> Even integers are multiples of 2.
> An even integer is a multiple of 2.

Both of these can be formalized as

$$(\forall\, x : \mathit{even}.x : \mathit{mult}.x.2).$$

This formalization is consistent with the following convention that we use.

Convention: If a *free variable* appears in a predicate expression, we assume that the entire expression is implicitly universally quantified.

We can substitute the operator $\exists$, read "there exists", for phrases such as

> *for some*, *some*, *exists*, *there are*, *there is*, and *at least one*.

Consider the following examples:

> Some even number is divisible by 3: $(\exists\, i : \mathit{even}.i : i \;\mathbf{mod}\; 3 = 0)$
> There is cat that does not have two eyes: $(\exists\, x : \mathit{cat}.x : \neg \mathit{has\_two}.x.\mathrm{eyes})$

As we have seen above, formalizing an English statement in predicate logic may require us to define predicate symbols and other functions to capture the desired relationship among variables. For example, the $\mathit{has\_two}\,.x.y$, $\mathit{cat}.x$, $\mathit{even}.x$, and $\mathit{mult}.x.n$ functions we defined above.

As another example consider the English statement "every graduate student took one of Professor Cunningham's classes and passed one theory class".

To formalize this statement we might introduce several predicates:

$\mathit{taken}.s.c$: Student s has completed class c.
$\mathit{passed}.s.c$: Student s received a passing grade in class c.
$\mathit{grad}.s$: Student s is a graduate student.
$\mathit{cunn}.c$: Class c is taught by Professor Cunningham.
$\mathit{theory}.c$: Class c is a theory class.

Using the above predicates, we can translate the English sentence to the following:

$$(\forall\, s : \mathit{grad}.s : (\exists\, c, c' :: \mathit{cunn}.c \wedge \mathit{taken}.s.c \wedge \mathit{theory}.c' \wedge \mathit{passed}.s.c'))$$

Alternatively, we can translate the statement to the following:

$$(\exists\, c, c' :: (\forall\, s : \mathit{grad}.s : \mathit{cunn}.c \wedge \mathit{taken}.s.c \wedge \mathit{theory}.c' \wedge \mathit{passed}.s.c'))$$

These two predicates have different meanings. The first says that every student took a class from Cunningham and passed a class in theory. The second says that all students took the same class from Cunningham and passed the same theory class. The English statement is ambiguous!

To deal formally with students and classes, we would need axioms and theorems that capture the desired properties of student transcripts. For example, we likely would want an axiom relating courses *passed* to courses *taken*:

$$\mathit{passed}.s.c \Rightarrow \mathit{taken}.s.c$$

THE AXIOMS AND IMPORTANT THEOREMS

Equivalence ($\equiv$) “equivales”

The equivalence operator $\equiv$ has lower binding power than the arithmetic and relational operators.

1. **Ax:** $((P \equiv Q) \equiv R) \equiv (P \equiv (Q \equiv R))$ $\equiv$ associativity
2. **Ax:** $(P \equiv Q) \equiv (Q \equiv P)$ $\equiv$ symmetry (commutativity)
3. **Ax:** $P \equiv \mathit{true} \equiv P$ $\equiv$ identity element (unit)
4. **Th:** $P \equiv P$ $\equiv$ reflexivity
5. **Th:** true

Disjunction ($\vee$) “or”

The disjunction operator $\vee$ has higher binding power than $\equiv$, but lower power than arithmetic and relational operators.

6. **Ax:** $(P \vee Q) \vee R \equiv P \vee (Q \vee R)$ $\vee$ associativity
7. **Ax:** $P \vee Q \equiv Q \vee P$ $\vee$ symmetry (commutativity)
8. **Ax:** $P \vee P \equiv P$ $\vee$ idempotence
9. **Ax:** $P \vee (Q \equiv R) \equiv P \vee Q \equiv P \vee R$ distribution of $\vee$ over $\equiv$
10. **Th:** $P \vee (Q \vee R) \equiv (P \vee Q) \vee (P \vee R)$ distribution of $\vee$ over itself
11. **Th:** $P \vee \mathit{true} \equiv \mathit{true}$ $\vee$ zero element

Conjunction ($\wedge$) “and”

The conjunction operator $\wedge$ has the same binding power as $\vee$.

12. **Ax:** $P \wedge Q \equiv P \equiv Q \equiv P \vee Q$ golden rule
13. **Th:** $P \wedge Q \equiv Q \wedge P$ $\wedge$ symmetry (commutativity)
14. **Th:** $(P \wedge Q) \wedge R \equiv P \wedge (Q \wedge R)$ $\wedge$ associativity
15. **Th:** $P \wedge P \equiv P$ $\wedge$ idempotence
16. **Th:** $P \wedge \mathit{true} \equiv P$ $\wedge$ identity element (unit)
17. **Th:** $P \wedge (Q \wedge R) \equiv (P \wedge Q) \wedge (P \wedge R)$ distribution of $\wedge$ over itself

18. **Th:** $P \wedge (Q \equiv R) \equiv P \wedge Q \equiv P \wedge R \equiv P$

19. **Th:** $P \wedge (Q \equiv R \equiv S) \equiv P \wedge Q \equiv P \wedge R \equiv P \wedge S$ distribution of $\wedge$ over $\equiv \equiv$

20. **Th:** $P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$ distribution of $\wedge$ over $\vee$

21. **Th:** $P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$ distribution of $\vee$ over $\wedge$

22. **Th:** $P \vee (P \wedge Q) \equiv P$ absorption

23. **Th:** $P \wedge (P \vee Q) \equiv P$ absorption

Implication ($\Rightarrow$) "implies"

The implication operator $\Rightarrow$ has higher binding power then $\equiv$, but lower power than $\vee$.

24. **Ax:** $P \Rightarrow Q \equiv P \vee Q \equiv Q$ implication

25. **Th:** $P \Rightarrow P$ $\Rightarrow$ reflexivity

26. **Th:** $P \Rightarrow \mathit{true}$ true consequent (right zero of $\Rightarrow$)

27. **Th:** $P \wedge Q \Rightarrow P$ conjunct simplification (weakening)

28. **Th:** $P \Rightarrow P \vee Q$ disjunct addition (weakening)

29. **Th:** $\mathit{true} \Rightarrow P \equiv P$ true antecedent (left identity of $\Rightarrow$)

30. **Th:** $P \Rightarrow Q \equiv P \wedge Q \equiv P$ $\Rightarrow$ to $\wedge$ connection

31. **Th:** $P \Rightarrow (Q \Rightarrow R) \equiv P \wedge Q \Rightarrow R$ antecedent importation

32. **Th:** $P \wedge (P \Rightarrow Q) \equiv P \wedge Q$ $\Rightarrow$ elimination

33. **Th:** $(P \Rightarrow Q) \wedge (Q \Rightarrow R) \Rightarrow (P \Rightarrow R)$ $\Rightarrow$ transitivity

34. **Th:** $(P \Rightarrow Q) \vee (Q \Rightarrow R)$

35. **Th:** $(P \equiv Q) \equiv (P \Rightarrow Q) \wedge (Q \Rightarrow P)$ mutual implication

Consequence ($\Leftarrow$) "follows from"

The consequence operator $\Leftarrow$ has the same binding power as $\Rightarrow$.

36. **Ax:** $P \Leftarrow Q \equiv P \vee Q \equiv P$ consequence

37. **Th:** $(P \Leftarrow Q) \equiv (Q \Rightarrow P)$ $\Leftarrow$ to $\Rightarrow$ connection

Negation ($\neg$) “not”

The negation operator $\neg$is a unary operator; it has higher binding power than $\vee$, but lower power than the arithmetic and relational operators.

38. **Ax:** $\neg(P \equiv Q) \equiv \neg P \equiv Q$ — negation
39. **Ax:** $P \vee \neg P$ — excluded middle
40. **Th:** $\neg P \equiv Q \equiv P \equiv \neg Q$
41. **Th:** $\neg\neg P \equiv P$ — double negation (inverse of $\neg$)
42. **Th:** $\neg P \vee Q \equiv P \vee Q \equiv Q$ — $\neg$ to $\vee$ connection
43. **Th:** $\neg P \wedge \neg Q \equiv \neg(P \vee Q)$ — de Morgan
44. **Th:** $\neg P \vee \neg Q \equiv \neg(P \wedge Q)$ — de Morgan
45. **Th:** $(P \wedge \neg Q) \vee Q \equiv P \vee Q$ — complement
46. **Th:** $(P \vee \neg Q) \wedge Q \equiv P \wedge Q$ — complement
47. **Th:** $\neg P \vee Q \equiv P \Rightarrow Q$ — $\Rightarrow$ to $\neg$ connection
48. **Th:** $P \Rightarrow Q \equiv \neg Q \Rightarrow \neg P$ — contraposition
49. **Th:** $(P \wedge Q \Rightarrow R) \equiv (P \Rightarrow \neg Q \vee R)$ — shuffle (shunting)
50. **Th:** $P \vee \neg \mathit{true} \equiv P$ — $\vee$ identity element (unit)
51. **Ax:** $\neg \mathit{true} \equiv \mathit{false}$ — definition of *false*
52. **Th:** $P \vee \mathit{false} \equiv P$ — $\vee$ identity element (unit)
53. **Th:** $\neg \mathit{false} \equiv \mathit{true}$
54. **Th:** $P \wedge \mathit{false} \equiv \mathit{false}$ — $\wedge$ zero element
55. **Th:** $P \equiv \neg P \equiv \mathit{false}$
56. **Th:** $\mathit{false} \Rightarrow P$ — false antecedent
57. **Th:** $P \wedge \neg P \equiv \mathit{false}$ — contradiction

Discrepancy ($\not\equiv$) “differs from”

The discrepancy operator $\not\equiv$ has the same binding power as $\equiv$.

58. **Ax:** $((P \not\equiv Q) \not\equiv R) \equiv (P \not\equiv (Q \not\equiv R))$ — $\not\equiv$ associativity
59. **Ax:** $P \not\equiv Q \equiv Q \not\equiv P$ — $\not\equiv$ symmetry (commutativity)

60. **Ax:** $P \equiv (Q \not\equiv R) \equiv (P \equiv Q) \not\equiv R$ association of $\not\equiv$ with $\equiv$

61. **Ax:** $P \not\equiv Q \equiv \neg(P \equiv Q)$

62. **Th:** $P \not\equiv \mathit{false} \equiv P$

63. **Th:** $P \wedge (Q \not\equiv R) \equiv P \wedge Q \not\equiv P \wedge R$ distribution of $\wedge$ over $\not\equiv$

Leibniz

64. **Ax:** $e = f \Rightarrow E(z := e) = E(z := f)$ Leibniz

Universal Quantification ($\forall$) "for all"

Universal quantification is the name given to the quantified version of $\wedge$. Conjunction is associative, symmetric, and idempotent; it has identity element *true* and zero element *false*.

65. **Ax:** : $(\forall i : R : P \wedge Q) \;\equiv\; (\forall i : R : P) \wedge (\forall i : R : Q)$ term splitting (or joining)

66. **Ax:** : $(\forall i : P \vee Q : T) \;\equiv\; (\forall i : P : T) \wedge (\forall i : Q : T)$ range splitting (or joining)

67. **Ax:** If R is nonempty and $\neg \mathit{occurs}(\text{'}i\text{'}, \text{'}Z\text{'})$,
$(\forall i : R : Z \wedge T) \;\equiv\; Z \wedge (\forall i : R : T)$ distribution of $\wedge$ over $\forall$

68. **Ax:** If R is nonempty and $\neg \mathit{occurs}(\text{'}i\text{'}, \text{'}Z\text{'})$,
$(\forall i : R : Z \vee T) \;\equiv\; Z \vee (\forall i : R : T)$ distribution of $\vee$ over $\forall$

69. **Ax:** $(\forall i : \mathit{false} : T) \;\equiv\; \mathit{true}$ empty range

70. **Ax:** If R is nonempty and $\neg \mathit{occurs}(\text{'}i\text{'}, \text{'}Z\text{'})$, $(\forall i : R : Z) \;\equiv\; Z$ constant term

71. **Ax:** If $\neg \mathit{occurs}(\text{'}j\text{'}, \text{'}R,T\text{'})$ and f is an invertible function,
$(\forall i : R : T) \;\equiv\; (\forall j : R(i := f.j) : T(i := f.j))$ dummy transformation

72. **Ax:** If $\neg \mathit{occurs}(\text{'}j\text{'}, \text{'}R\text{'})$,
$(\forall i : R : (\forall j : Q : T)) \;\equiv\; (\forall i, j : R \wedge Q : T)$ nesting (or unnesting)

73. **Ax:** If $\neg \mathit{occurs}(\text{'}i\text{'}, \text{'}Z\text{'})$, $(\forall i : i = Z : T) \;\equiv\; T(i := Z)$ one point

74. **Ax:** If $\neg \mathit{occurs}(\text{'}j\text{'}, \text{'}P\text{'})$ and $\neg \mathit{occurs}(\text{'}i\text{'}, \text{'}Q\text{'})$,
$(\forall i : P : (\forall j : Q : T)) \;\equiv\; (\forall j : Q : (\forall i : P : T))$ interchange

75. **Ax:** $(\forall i : R : T) \;\equiv\; (\forall i :: \neg R \vee T)$ trading

76. **Th:** If $\neg \mathit{occurs}(\text{'}j\text{'}, \text{'}R,T\text{'})$,
$(\forall i : R : T) \;\equiv\; (\forall j : R(i := j) : T(i := j))$ dummy renaming

77. **Th:** If $\neg \mathit{occurs}(\text{'}i\text{'}, \text{'}Z\text{'})$,
$(\forall i : R : Z \vee T) \;\equiv\; Z \vee (\forall i : R : T)$ distribution of $\vee$ over $\forall$

78. **Th:** If $\neg \mathit{occurs}(\text{'}i\text{'}, \text{'}Z\text{'})$,
$(\forall i : R : Z \Rightarrow T) \;\equiv\; Z \Rightarrow (\forall i : R : T)$ distribution of ⇒ over ∀

79. **Th:** $(\forall i : R \wedge S : T) \;\equiv\; (\forall i : R : S \Rightarrow T)$ trading

80. **Th:** $(\forall i :: T) \Rightarrow T(i := E)$ instantiation

Existential Quantification (∃) "there exists"

Existential quantification is the name given to the quantified version of ∨. Disjunction is associative, symmetric, and idempotent; it has identity element *false* and zero element *true*.

81. **Ax:** : $(\exists i : R : P \vee Q) \;\equiv\; (\exists i : R : P) \vee (\exists i : R : Q)$ term splitting (or joining)

82. **Ax:** : $(\exists i : P \vee Q : T) \;\equiv\; (\exists i : P : T) \vee (\exists i : Q : T)$ range splitting (or joining)

83. **Ax:** If R is nonempty and $\neg \mathit{occurs}(\text{'}i\text{'}, \text{'}Z\text{'})$,
$(\exists i : R : Z \vee T) \;\equiv\; Z \vee (\exists i : R : T)$ distribution of ∨ over ∃

84. **Ax:** If R is nonempty and $\neg \mathit{occurs}(\text{'}i\text{'}, \text{'}Z\text{'})$,
$(\exists i : R : Z \wedge T) \;\equiv\; Z \wedge (\exists i : R : T)$ distribution of ∧ over ∃

85. **Ax:** $(\exists i : \mathit{false} : T) \;\equiv\; \mathit{false}$ empty range

86. **Ax:** If R is nonempty and $\neg \mathit{occurs}(\text{'}i\text{'}, \text{'}Z\text{'})$, $(\exists i : R : Z) \;\equiv\; Z$ constant term

87. **Ax:** If $\neg \mathit{occurs}(\text{'}j\text{'}, \text{'}R,T\text{'})$ and f is an invertible function,
$(\exists i : R : T) \;\equiv\; (\exists j : R(i := f.j) : T(i := f.j))$ dummy transformation

88. **Ax:** If $\neg \mathit{occurs}(\text{'}j\text{'}, \text{'}R\text{'})$,
$(\exists i : R : (\exists j : Q : T)) \;\equiv\; (\exists i, j : R \wedge Q : T)$ nesting (or unnesting)

89. **Ax:** If $\neg \mathit{occurs}(\text{'}i\text{'}, \text{'}Z\text{'})$, $(\exists i : i = Z : T) \;\equiv\; T(i := Z)$ one point

90. **Ax:** If $\neg \mathit{occurs}(\text{'}j\text{'}, \text{'}P\text{'})$ and $\neg \mathit{occurs}(\text{'}i\text{'}, \text{'}Q\text{'})$,
$(\exists i : P : (\exists j : Q : T)) \;\equiv\; (\exists j : Q : (\exists i : P : T))$ interchange

91. **Ax:** $(\exists i : R : T) \;\equiv\; \neg(\forall i : R : \neg T)$ generalized de Morgan

92. **Th:** $(\forall i : R : T) \;\equiv\; \neg(\exists i : R : \neg T)$ generalized de Morgan

93. **Th:** If $\neg \mathit{occurs}(\text{'}j\text{'}, \text{'}R,T\text{'})$,
$(\exists i : R : T) \;\equiv\; (\exists j : R(i := j) : T(i := j))$ dummy renaming

94. **Th:** If $\neg \mathit{occurs}(\text{'}i\text{'}, \text{'}Z\text{'})$,
$(\exists i : R : Z \wedge T) \;\equiv\; Z \wedge (\exists i : R : T)$ distribution of ∧ over ∃

95. **Th:** $(\exists i : R : T) \;\equiv\; (\exists i :: R \wedge T)$ trading

96. **Th:** $T(i := E) \Rightarrow (\exists i :: T)$ instantiation

Summation (Σ) "sum of"

Summation is the name given to the quantified version of $+$. A summation is only defined for a finite range. The addition ($+$) operation is associative and symmetric but not idempotent; it has identity element 0 but no zero element.

Below let E and F be integer expressions.

97. **Ax:** $(\Sigma\, i : R : E + F) \;=\; (\Sigma\, i : R : E) + (\Sigma\, i : R : F)$ term splitting (or joining)

98. **Ax:** $(\Sigma\, i : P \vee Q : E) + (\Sigma\, i : P \wedge Q : E) \;=\; (\Sigma\, i : P : E) + (\Sigma\, i : Q : E)$ range splitting

99. **Ax:** $(\Sigma\, i : \mathit{false} : E) \;=\; 0$ empty range

100. **Ax:** If $\neg$*occurs*(*'j'*, *'R,E'*) and f is an invertible function,
$(\Sigma\, i : R : E) \;=\; (\Sigma\, j : R(i := f.j) : E(i := f.j))$ dummy transformation

101. **Ax:** If $\neg$*occurs*(*'j'*, *'P'*),
$(\Sigma\, i : P : (\Sigma\, j : Q : E)) \;=\; (\Sigma\, i, j : P \wedge Q : E)$ nesting (or unnesting)

102. **Ax:** If $\neg$*occurs*(*'i'*, *'Z'*),
$(\Sigma\, i : i = Z : E) \;=\; E(i := Z)$ one point

103. **Ax:** If $\neg$*occurs*(*'j'*, *'P'*) and $\neg$*occurs*(*'i'*, *'Q'*),
$(\Sigma\, i : P : (\Sigma\, j : Q : E)) \;=\; (\Sigma\, j : Q : (\Sigma\, i : P : E))$ interchange

104. **Th:** If $\neg$*occurs*(*'j'*, *'R,E'*),
$(\Sigma\, i : R : E) \;=\; (\Sigma\, j : R(i := j) : E(i := j))$ dummy renaming

105. **Th:** If R is nonempty and $\neg$*occurs*(*'i'*, *'Z'*),
$(\Sigma\, i : R : Z * E) \;=\; Z * (\Sigma\, i : R : E)$ distribution of $*$ over Σ

Product (Π) "product of"

Product is the name given to the quantified version of $*$. A product is only defined for a finite range. The multiplication ($*$) operation is associative and symmetric but not idempotent; it has identity element 1 and zero element 0.

Below let E and F be integer expressions.

106. **Ax:** $(\Pi\, i : R : E * F) \;=\; (\Pi\, i : R : E) * (\Pi\, i : R : F)$ term splitting (or joining)

107. **Ax:** $(\Pi\, i : P \vee Q : E) * (\Pi\, i : P \wedge Q : E) \;=\; (\Pi\, i : P : E) * (\Pi\, i : Q : E)$ range splitting

108. **Ax:** $(\Pi\, i : \mathit{false} : E) \;=\; 1$ empty range

109. **Ax:** If $\neg$*occurs*(*'j'*, *'R,E'*) and f is an invertible function,
$(\Pi\, i : R : E) \;=\; (\Pi\, j : R(i := f.j) : E(i := f.j))$ dummy transformation

110. **Ax:** If $\neg$*occurs*(*'j'*, *'P'*),
$(\Pi\, i : P : (\Pi\, j : Q : E)) \;=\; (\Pi\, i, j : P \wedge Q : E)$ nesting (or unnesting)

111. **Ax:** If $\neg \mathit{occurs}('i', 'Z')$, $(\Pi\, i : i = Z : E) \;=\; E(i := Z)$ one point

112. **Ax:** If $\neg \mathit{occurs}('j', 'P')$ and $\neg \mathit{occurs}('i', 'Q')$,
$(\Pi\, i : P : (\Pi\, j : Q : E)) \;=\; (\Pi\, j : Q : (\Pi\, i : P : E))$ interchange

113. **Th:** If $\neg \mathit{occurs}('j', 'R,E')$,
$(\Pi\, i : R : E) \;=\; (\Pi\, j : R(i := j) : E(i := j))$ dummy renaming

Maximization (MAX) “maximum of”

Maximization is the name given to the quantified version of **max**. The operation **max** is associative, symmetric, and idempotent; it has identity element $-\infty$ and and zero element $+\infty$.

Below let E, F, and Z be integer expressions.

114. **Ax:** $(\mathbf{MAX}\ i : R : E\ \mathbf{max}\ F) \;=\; (\mathbf{MAX}\ i : R : E)\ \mathbf{max}\ (\mathbf{MAX}\ i : R : F)$
term splitting (or joining)

115. **Ax:** $(\mathbf{MAX}\ i : P \vee Q : E) \;=\; (\mathbf{MAX}\ i : P : E)\ \mathbf{max}\ (\mathbf{MAX}\ i : Q : E)$
range splitting (or joining)

116. **Ax:** If R is nonempty and $\neg \mathit{occurs}('i', 'Z')$, distribution **max** over **MAX**
$(\mathbf{MAX}\ i : R : Z\ \mathbf{max}\ E) \;=\; Z\ \mathbf{max}\ (\mathbf{MAX}\ i : R : E)$

117. **Ax:** $(\mathbf{MAX}\ i : \mathit{false} : E) \;=\; -\infty$ empty range

118. **Ax:** If R is nonempty and $\neg \mathit{occurs}('i', 'Z')$, $(\mathbf{MAX}\ i : R : Z) \;=\; Z$ constant term

119. **Ax:** If $\neg \mathit{occurs}('j', 'R,E')$ and f is an invertible function,
$(\mathbf{MAX}\ i : R : E) \;=\; (\mathbf{MAX}\ j : R(i := f.j) : E(i := f.j))$ dummy transformation

120. **Ax:** If $\neg \mathit{occurs}('j', 'P')$, nesting (or unnesting)
$(\mathbf{MAX}\ i : P : (\mathbf{MAX}\ j : Q : E)) \;=\; (\mathbf{MAX}\ i, j : P \wedge Q : E)$

121. **Ax:** If $\neg \mathit{occurs}('i', 'Z')$, $(\mathbf{MAX}\ i : i = Z : E) \;=\; E(i := Z)$ one point

122. **Ax:** If $\neg \mathit{occurs}('j', 'P')$ and $\neg \mathit{occurs}('i', 'Q')$, interchange
$(\mathbf{MAX}\ i : P : (\mathbf{MAX}\ j : Q : E)) \;=\; (\mathbf{MAX}\ j : Q : (\mathbf{MAX}\ i : P : E))$

123. **Th:** If $\neg \mathit{occurs}('j', 'R,E')$,
$(\mathbf{MAX}\ i : R : E) \;=\; (\mathbf{MAX}\ j : R(i := j) : E(i := j))$ dummy renaming

124. **Th:** If R is nonempty and $\neg \mathit{occurs}('i', 'Z')$,
$(\mathbf{MAX}\ i : R : Z + E) \;=\; Z + (\mathbf{MAX}\ i : R : E)$ distribution of + over **MAX**

125. **Th:** If R is nonempty, $\neg \mathit{occurs}('i', 'Z')$, and $Z \geq 0$,
$(\mathbf{MAX}\ i : R : Z * E) \;=\; Z * (\mathbf{MAX}\ i : R : E)$ distribution of $*$ over **MAX**

126. **Th:** If R is nonempty and $\neg \mathit{occurs}('i', 'Z')$, distribution **min** over **MAX**
$Z\ \mathbf{min}\ (\mathbf{MAX}\ i : R : E) \;=\; (\mathbf{MAX}\ i : R : Z\ \mathbf{min}\ E)$

127. **Ax:** If R is nonempty, $f.x = (\mathbf{MAX}\ i : R : f.i) \;\equiv\; R(i := x) \wedge (\forall\, i : R : f.i \leq f.x)$

Minimization (MIN) “minimum of”

Minimization is the name given to the quantified version of **min**. The operation **min** is associative, symmetric, and idempotent; it has identity element $+\infty$ and and zero element $-\infty$.

Below let E, F, and Z be integer expressions.

128. **Ax:** $(\textbf{MIN}\ i : R : E\ \textbf{min}\ F)\ =\ (\textbf{MIN}\ i : R : E)\ \textbf{min}\ (\textbf{MIN}\ i : R : F)$
term splitting (or joining)

129. **Ax:** $(\textbf{MIN}\ i : P \vee Q : E)\ =\ (\textbf{MIN}\ i : P : E)\ \textbf{min}\ (\textbf{MIN}\ i : Q : E)$
range splitting (or joining)

130. **Ax:** If R is nonempty and $\neg$*occurs*(’*i*’,’*Z*’), distribution of **min** over **MIN**
$(\textbf{MIN}\ i : R : Z\ \textbf{min}\ E)\ =\ Z\ \textbf{min}\ (\textbf{MIN}\ i : R : E)$

131. **Ax:** $(\textbf{MIN}\ i : false : E)\ =\ +\infty$ empty range

132. **Ax:** If R is nonempty and $\neg$*occurs*(’*i*’,’*Z*’), $(\textbf{MIN}\ i : R : Z)\ =\ Z$ constant term

133. **Ax:** If $\neg$*occurs*(’*j*’,’*R,E*’) and f is an invertible function,
$(\textbf{MIN}\ i : R : E)\ =\ (\textbf{MIN}\ j : R(i := f.j) : E(i := f.j))$ dummy transformation

134. **Ax:** If $\neg$*occurs*(’*j*’,’*P*’), nesting (or unnesting)
$(\textbf{MIN}\ i : P : (\textbf{MIN}\ j : Q : E))\ =\ (\textbf{MIN}\ i,j : P \wedge Q : E)$

135. **Ax:** If $\neg$*occurs*(’*i*’,’*Z*’), $(\textbf{MIN}\ i : i = Z : E)\ =\ E(i := Z)$ one point

136. **Ax:** If $\neg$*occurs*(’*j*’,’*P*’) and $\neg$*occurs*(’*i*’,’*Q*’), interchange
$(\textbf{MIN}\ i : P : (\textbf{MIN}\ j : Q : E))\ =\ (\textbf{MIN}\ j : Q : (\textbf{MIN}\ i : P : E))$

137. **Th:** If $\neg$*occurs*(’*j*’,’*R,E*’),
$(\textbf{MIN}\ i : R : E)\ =\ (\textbf{MIN}\ j : R(i := j) : E(i := j))$ dummy renaming

138. **Th:** If R is nonempty and $\neg$*occurs*(’*i*’,’*Z*’),
$(\textbf{MIN}\ i : R : Z + E)\ =\ Z + (\textbf{MIN}\ i : R : E)$ distribution of + over **MIN**

139. **Th:** If R is nonempty, $\neg$*occurs*(’*i*’,’*Z*’), and $Z \geq 0$,
$(\textbf{MIN}\ i : R : Z * E)\ =\ Z * (\textbf{MIN}\ i : R : E)$ distribution of $*$ over **MIN**

140. **Th:** If R is nonempty and $\neg$*occurs*(’*i*’,’*Z*’), distribution **max** over **MIN**
$Z\ \textbf{max}\ (\textbf{MIN}\ i : R : E)\ =\ (\textbf{MIN}\ i : R : Z\ \textbf{max}\ E)$

141. **Ax:** $(\textbf{MIN}\ i : R : -E)\ =\ -(\textbf{MAX}\ i : R : E)$

142. **Ax:** If R is nonempty, $f.x = (\textbf{MIN}\ i : R : f.i)\ \equiv\ R(i := x) \wedge (\forall\, i : R : f.i \geq f.x)$

Numerical Quantification (#) “number of”

Let operation # be defined such that $\#.true = 1$ and $\#.false = 0$.

Numerical quantification is the quantified version of the unary operator #. A numerical quantification is only defined for a finite range.

143. **Ax:** $(\# i : R : T) \ = \ (\Sigma i : R \wedge T : 1)$ numerical quantification

144. **Ax:** $(\# i : R : T) \ = \ (\Sigma i : R : \#.T)$ numerical quantification

145. **Th:** $(\# i : P \vee Q : T) + (\# i : P \wedge Q : T) \ = \ (\# i : P : T) + (\# i : Q : T)$ range splitting

146. **Th:** $(\# i : false : T) \ = \ 0$ empty range

147. **Th:** If $\neg$*occurs*(’j’,’R,T’) and f is an invertible function,
$(\# i : R : T) \ = \ (\# j : R(i := f.j) : T(i := f.j))$ dummy transformation

148. **Th:** If $\neg$*occurs*(’i’,’Z’), $(\# i : i = Z : T) \ = \ \#.(T(i := Z))$ one point

149. **Th:** If $\neg$*occurs*(’j’,’R,T’),
$(\# i : R : T) \ = \ (\# j : R(i := j) : T(i := j))$ dummy renaming

150. **Th:** If $\neg$*occurs*(’i’,’Z’), $(\Sigma i : R \wedge T : Z) \ = \ Z * (\# i : R : T)$

151. **Ax:** $(\exists i : R : T) \ \equiv \ (\# i : R : T) \geq 1$

152. **Ax:** $(\forall i : R : T) \ \equiv \ (\# i : R : T) = (\# i : R : true)$