Chapter 2

The predicate calculus

2.1 Its relevance

The pre- and postconditions of specifications are predicate calculus formulae. And some refinement rules have formulae attached as provisos, meaning that the rule is applicable only if its attached formula is true — which may require a proof. Thus we use the predicate calculus in two ways: for describing, and for reasoning.

Predicate calculus was developed by logicians well before the appearance of computers. With it, they hoped to formalise human reasoning about mathematics at least. The truth or falsity of any conjecture was to be decided as follows:

1. Express the conjecture as a formula $\mathcal{A}$. (That requires a precise notion of the meaning of formulae.)
2. Using a precise system of proof, given beforehand, either prove $\mathcal{A}$ or its negation $\neg\mathcal{A}$.

The system of proof (based on axioms and inference rules) was designed so that there could never be any doubt about whether a text was a proof or not. Finding a proof, however, remained as much a problem as before.

In fact, it became harder to find proofs — and not only because incorrect ones were newly excluded! The rigorous rules each expressed very small reasoning steps, and so proofs required very many of them. But in a theoretical sense that did not matter: every true formula could be proved in that way.

Predicate calculus is of practical concern to computer scientists, however. To use it effectively, we must avoid long proofs. We do that in three ways. First, we choose our laws of program refinement so that they generate few proof obligations. For example, we do not include feasibility checks at every stage, because infeasible programs cannot lead to (incorrect) code — they lead to no code at all. They lose time, but not human life.

Second, we use routinely the more advanced techniques of logical argument (proof by contradiction, etc.) which have themselves been justified formally by others. We will not justify them ourselves.

Finally, in each program to be developed we look for suitable notation of *our* devising, appropriate to the characteristics of the problem. We might assume some properties of the structures involved, calling them 'obvious'; others we might prove.

In summary: We do not use or advocate any particular system of formal logical reasoning with the predicate calculus (axiomatic, natural deduction, tableaux, etc.). Our use of the predicate calculus is based on familiarity (eventually!) with a number of predicate laws, usually equalities between formulae, which are used to reduce a complex formula to a simple one. That is how other calculi in mathematics are employed; we only do the same.

2.2 Terms

Terms (also called expressions) are built from variables, constants, and functions. Thus x on its own is a term (it is a variable); and 1 is a term (it is a constant); and $x+1$ is a term (it is formed by applying the function $+$ to the two terms x and 1). A state, which maps variables to values (recall Chapter 1), determines the values of terms: one speaks of a term having some value in a state. In a state that maps x to three, the term x has the value three (trivially), and 0 has the value zero (in every state, in fact: that is why it is called a constant), and $x+1$ has the value four.

Our *variables* will have short lower-case *italic* names, drawn from the Roman alphabet.

Our *constants* will have their usual mathematical names, like 0 and π. (The real number constants e and i will not cause trouble.)

Our *functions* will have their usual mathematical names too, like square root $\sqrt{}$, plus $+$, and factorial $!$. Some of those take one argument ($\sqrt{}$ and !), some take two ($+$), and the position of the arguments can vary: sometimes the function is written before its argument ($\sqrt{}$), sometimes between its arguments ($+$), and sometimes after its argument (!). The number of arguments a function takes is called its *arity*.

We often need to introduce new functions, of our own, just for a particular problem. For those, the syntax is more regular: they will have short lower-case $\mathsf{sanserif}$ names, in the Roman alphabet. Their arguments follow them, separated by spaces. For uniformity, we use that convention even for the mathematical functions log, sin, etc.

Terms are made from all the above. A *term* is either

1. a variable;
2. a constant; or
3. a function applied to the correct number of other terms, depending on its arity.

Figure 2.1 lists some terms.

$$0$$
$$x$$
$$x+1$$
$$\mathsf{log}\, x$$
$$\mathsf{sin}(\pi/2)$$
$$(a+b)\times 3!$$

Figure 2.1 Some terms

$$\mathsf{false}$$
$$1 < (a \div 2)$$
$$(x+1) = 7$$
$$\mathsf{even}\, 6$$
$$\pi \in \mathbb{R}$$

Figure 2.2 Some simple formulae

2.3 Simple formulae

Simple formulae[1] are built from terms and predicate symbols. The best-known predicate symbols represent the binary relations from arithmetic: $<$, $=$, $\leq$ etc. Like functions, predicates have an arity; for binary relations, the arity is two. Again like functions, predicates are applied to terms.

Unlike functions, a predicate applied to (the correct number of) terms is not another term: it is a *simple formula*. Simple formulae do not have general values like terms; instead, they take only the values true and false.

For conventional predicates (like binary relations) we use the usual notation. Predicates that we introduce ourselves will be short Roman $\mathsf{sanserif}$ names, and their arguments will follow them, separated by spaces (as for our functions).

Finally, there are the two constant predicates true and false. In every state, the first is true and the second is false.

Figure 2.2 lists some simple formulae.

[1] They are called *atomic* formulae in the logic literature.

$\mathcal{A}$	$\mathcal{B}$	$\mathcal{A} \wedge \mathcal{B}$
true	true	true
true	false	false
false	true	false
false	false	false

$\mathcal{A}$	$\mathcal{B}$	$\mathcal{A} \vee \mathcal{B}$
true	true	true
true	false	true
false	true	true
false	false	false

$\mathcal{A}$	$\mathcal{B}$	$\mathcal{A} \Rightarrow \mathcal{B}$
true	true	true
true	false	false
false	true	true
false	false	true

$\mathcal{A}$	$\mathcal{B}$	$\mathcal{A} \Leftrightarrow \mathcal{B}$
true	true	true
true	false	false
false	true	false
false	false	true

$\mathcal{A}$	$\neg\mathcal{A}$
true	false
false	true

Figure 2.3 Truth tables for propositional connectives

2.4 Propositional formulae

Propositional formulae are built from simple formulae, using propositional connectives. The connectives are $\wedge$ (and), $\vee$ (or), $\neg$ (not), $\Rightarrow$ (implies), and $\Leftrightarrow$ (if and only if, or iff). (As nouns, they are conjunction, disjunction, negation, implication and equivalence.) Except for $\neg$, all have two arguments, written on either side; the single argument of $\neg$ is written after it.

Like simple formulae, propositional formulae are either true or false, once given a state. If, for example, $\mathcal{A}$ and $\mathcal{B}$ are propositional formulae, then the propositional formula $\mathcal{A} \wedge \mathcal{B}$ is true exactly when both $\mathcal{A}$ *and* $\mathcal{B}$ are true. That is summarised in this table:

$\mathcal{A}$	$\mathcal{B}$	$\mathcal{A} \wedge \mathcal{B}$
true	true	true
true	false	false
false	true	false
false	false	false

A complete set of 'truth tables' for the five connectives is given in Figure 2.3. In a formula $\mathcal{A} \Rightarrow \mathcal{B}$, the subformula $\mathcal{A}$ is the *antecedent*, and $\mathcal{B}$ is the *consequent*.

Following convention, we allow the abbreviation $a < b < c$ (and similar) for the propositional formula $a < b \wedge b < c$.

Figure 2.4 gives some propositional formulae.

$$\begin{gathered}\mathsf{true}\\ x^2 = -1\\ (x \le y) \wedge (y \le x+1)\\ (x > 0) \Rightarrow (x + y \ne y)\\ (0 \le p < q) \Rightarrow (0 < q)\\ (n! = n) \Leftrightarrow (n = 1) \vee (n = 2)\end{gathered}$$

Figure 2.4 Some propositional formulae

2.5 Quantifiers

2.5.1 Universal quantification

A *universally quantified* formula is written

$$(\forall\, x \bullet \mathcal{A})\,,$$

where x is a variable, called the *bound* variable, and $\mathcal{A}$ is some other formula, called the *body*. It is true exactly when $\mathcal{A}$ is true for all values of x, where it is understood that we know the set from which those values of x are drawn (for example, the real numbers). We also allow a list of bound variables, as in $(\forall\, x, y \bullet \mathcal{A})$. There, the quantification is true exactly when the body is true for all values of those variables chosen independently. The order in the list does not affect the meaning.

Consider this parody of the distributive law from arithmetic:

$$a + (b \times c) \quad = \quad (a + b) \times (a + c)\ .$$

Although one would say informally 'that is false', it is in fact true in some states. (Map all three variables to one-third.)

But the quantified formula

$$(\forall\, a, b, c \bullet a + (b \times c) = (a + b) \times (a + c)) \tag{2.1}$$

is identically false, because it is not the case that the body is true for *all* values of a, b, and c.

Now consider the similar formula

$$(\forall\, b, c \bullet a + (b \times c) = (a + b) \times (a + c))\,, \tag{2.2}$$

in which we have quantified only b and c. It depends on a; and it is true when a is zero, and false otherwise.

2.5.2 Free and bound variables

Formula (2.2) depends on a, but not on b or c. Variable a is a *free* variable; variables b and c are not free, because they are bound by the quantifier $\forall$. In fact, variables b and c are just place-holders in that formula, indicating the positions at which all values are to be considered. Changing their names does not affect the formula (provided the new names do not conflict with existing ones). Thus

$$(\forall\, d, e \cdot a + (d \times e) = (a + d) \times (a + e))$$

has the same meaning as (2.2). Formula (2.1) has no free variables, since a, b, c are bound; it does not depend on the value of any variable.

In general, *bound* variables are those bound by a quantifier, as is x in $(\forall\, x \cdot \mathcal{A})$; all free occurrences of x in $\mathcal{A}$ itself become bound occurrences in the larger $(\forall\, x \cdot \mathcal{A})$.

Section A.2.1 further discusses free and bound variables.

2.5.3 Existential quantification

Existential quantification is used to express 'there exists'. An *existentially quantified* formula is written

$$(\exists\, x \cdot \mathcal{A})\,,$$

where x and $\mathcal{A}$ are as before. It is true exactly when there exists a value for x that makes $\mathcal{A}$ true. So the existentially quantified formula

$$(\exists\, a, b, c \cdot a + (b \times c) = (a + b) \times (a + c))$$

is true. Free occurrences of x in $\mathcal{A}$ are bound in $(\exists\, x \cdot \mathcal{A})$ just as they are in $(\forall\, x \cdot \mathcal{A})$.

2.5.4 Typed quantifications

A *typed quantification* indicates explicitly the set from which values for the bound variable are drawn. For example, let $\mathbb{Z}$ denote the set of all integers, and $\mathbb{N}$ the set of all natural numbers (non-negative integers). Then $(\exists\, x : \mathbb{Z} \cdot x < 0)$ is true, but $(\exists\, x : \mathbb{N} \cdot x < 0)$ is false (because 0 is the least natural number). In general, *typed* quantifications are written

$$(\forall\, x : T \cdot \mathcal{A}) \quad \text{and} \quad (\exists\, x : T \cdot \mathcal{A})\,,$$

where T denotes some set of values. The variable x then ranges over that set.

If we know beforehand the set from which values are drawn, we can use the simpler untyped quantifiers; the typing is then understood from context. But when several such sets are involved simultaneously, we use typed quantifiers.

$$\mathsf{true}$$
$$x \neq 3$$
$$y > 0 \Rightarrow y \neq 0$$
$$(\forall\, x : \mathbb{R} \cdot (\exists\, y : \mathbb{C} \cdot y^2 = x))$$
$$a \div b = c \Leftrightarrow (\exists\, r \cdot 0 \leq r < b \wedge a = b \times c + r)$$

Figure 2.5 Some general formulae

2.6 (General) formulae

Now we draw together all the above. A *formula* is any one of the following:

1. A simple formula.
2. $\neg\mathcal{A}$, where $\mathcal{A}$ is a formula.
3. $\mathcal{A} \wedge \mathcal{B}$, $\mathcal{A} \vee \mathcal{B}$, $\mathcal{A} \Rightarrow \mathcal{B}$, or $\mathcal{A} \Leftrightarrow \mathcal{B}$, where $\mathcal{A}$ and $\mathcal{B}$ are formulae.
4. $(\forall\, x : T \cdot \mathcal{A})$ or $(\exists\, x : T \cdot \mathcal{A})$, where x is a list of variables, T denotes a set, and $\mathcal{A}$ is a formula.

That definition allows nested quantifications, such as

$$(\forall\, a : \mathbb{R} \cdot (\exists\, b, c : \mathbb{R} \cdot a + (b \times c) = (a + b) \times (a + c)))$$

(which is true), and the application of propositional operators to quantifications, such as

$$x \neq 0 \Rightarrow (\exists\, y : \mathbb{Z} \cdot 0 \leq y \wedge y < x)\,,$$

true if x is a natural number.

Figure 2.5 gives some general formulae.

2.7 Operator precedence

Strictly speaking, a term like $2 + 3 \times 4$ is ambiguous: is its value fourteen or twenty? Such questions can be resolved by parentheses — $2 + (3 \times 4)$ *vs* $(2 + 3) \times 4$ — but they can be resolved also by general precedence rules. The usual rule from arithmetic is that $\times$ is done before $+$: we say that $\times$ has *higher precedence.*

We adopt all the usual precedence rules from arithmetic, adding to them that functions have highest precedence of all: thus $\sqrt{4} + 5$ is seven, not three. When several functions are used, the rightmost is applied first: thus $\mathsf{log}\,\mathsf{sin}(\pi/2)$ is zero.[2] We do not require parentheses around function arguments; but note that $\mathsf{sin}\,\pi/2$ is zero, whereas $\mathsf{sin}(\pi/2)$ is one.

[2]Without higher-order functions, the reverse does not make sense anyway.

In propositional formulae, the precedence is (highest) $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$ (lowest).

There is no need for precedence rules of quantifiers, because they are always written with enclosing parentheses $(\cdots)$ that give their scope.

2.8 Predicate calculation

2.8.1 Relations between formulae

The two (simple) formulae $x = y \Rightarrow x \neq z$ and $x = z \Rightarrow x \neq y$ are equivalent in this sense: in every state they are both true or both false together. In general, that two formulae $\mathcal{A}$ and $\mathcal{B}$ are *equivalent* is written $\mathcal{A} \equiv \mathcal{B}$, and means

In every state, $\mathcal{A}$ is true if and only if $\mathcal{B}$ is true .

That is indeed the same as saying 'in every state, $\mathcal{A} \Leftrightarrow \mathcal{B}$ is true'. But there is an important difference between $\equiv$ and $\Leftrightarrow$. The first is a relation between formulae: $\mathcal{A} \equiv \mathcal{B}$ is a statement about $\mathcal{A}$ and $\mathcal{B}$; it is not a formula itself. The second is a propositional connective: $\mathcal{A} \Leftrightarrow \mathcal{B}$ says nothing about formulae; rather it is a formula itself.

Here are two other relations between formulae. The statement $\mathcal{A} \Rrightarrow \mathcal{B}$ means

In every state, if $\mathcal{A}$ is true then $\mathcal{B}$ is true .

That is the same as 'in every state, $\mathcal{A} \Rightarrow \mathcal{B}$ is true'. And the statement $\mathcal{A} \Lleftarrow \mathcal{B}$ means

In every state, $\mathcal{A}$ is true if $\mathcal{B}$ is true .

It is the same as 'in every state, $\mathcal{B} \Rightarrow \mathcal{A}$ is true'. The relation $\Rrightarrow$ is known as *entailment*.

Those three relations are used to set out chains of reasoning like this one: for any formulae $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$,

$$
\begin{array}{ll}
 & (\mathcal{A} \Rightarrow \mathcal{C}) \vee (\mathcal{B} \Rightarrow \mathcal{C}) \\
\equiv & \text{“writing implication as disjunction”} \\
 & (\neg\mathcal{A} \vee \mathcal{C}) \vee (\neg\mathcal{B} \vee \mathcal{C}) \\
\equiv & \text{“associativity, commutativity of } \vee\text{”} \\
 & (\neg\mathcal{A} \vee \neg\mathcal{B}) \vee (\mathcal{C} \vee \mathcal{C}) \\
\equiv & \text{“de Morgan, idempotence of } \vee\text{”} \\
 & \neg(\mathcal{A} \wedge \mathcal{B}) \vee \mathcal{C} \\
\equiv & \text{“writing disjunction as implication”} \\
 & \mathcal{A} \wedge \mathcal{B} \Rightarrow \mathcal{C} \ .
\end{array}
$$

Each formula is related to the one before it by the relation $\equiv$, $\Rrightarrow$, or $\Lleftarrow$. And each step between formulae carries a decoration, a 'hint', suggesting why it is valid. The quotes “” separate the hints from the proof itself. They are not part of the proof; they are *about* the proof.

The relation $\equiv$ is *transitive*, which means that whenever both $\mathcal{A} \equiv \mathcal{B}$ and $\mathcal{B} \equiv \mathcal{C}$ (which we can write $\mathcal{A} \equiv \mathcal{B} \equiv \mathcal{C}$), then we have $\mathcal{A} \equiv \mathcal{C}$ too. That is why the chain of equivalences above establishes overall that the first formula is equivalent to the last:

$$(\mathcal{A} \Rightarrow \mathcal{C}) \vee (\mathcal{B} \Rightarrow \mathcal{C}) \quad \equiv \quad \mathcal{A} \wedge \mathcal{B} \Rightarrow \mathcal{C} \ .$$

The other relations $\Rrightarrow$ and $\Lleftarrow$ are transitive as well, but not if mixed together. Either can be mixed with $\equiv$, however; thus from $\mathcal{A} \equiv \mathcal{B} \Rrightarrow \mathcal{C}$ we still have $\mathcal{A} \Rrightarrow \mathcal{C}$. Finally, writing just $\Rrightarrow \mathcal{A}$ on its own means that $\mathcal{A}$ is true in every state.

2.8.2 Laws for calculation

To reason as above requires some knowledge of the laws to which one can appeal, like “associativity, commutativity of $\vee$”. Appendix A contains a collection of them. Each can be used to justify steps in a calculation, and often there are several that will do. One soon acquires favourites.

We do not present all those laws here; indeed, it will be some time before we need many of them. Where helpful, however, we refer to them directly. The reasoning above proved Predicate law A.36; here it is again, by numbers:

$$
\begin{array}{ll}
 & (\mathcal{A} \Rightarrow \mathcal{C}) \vee (\mathcal{B} \Rightarrow \mathcal{C}) \\
\equiv & \text{“Predicate law A.22”} \\
 & (\neg\mathcal{A} \vee \mathcal{C}) \vee (\neg\mathcal{B} \vee \mathcal{C}) \\
\equiv & \text{“Predicate laws A.3, A.5”} \\
 & (\neg\mathcal{A} \vee \neg\mathcal{B}) \vee (\mathcal{C} \vee \mathcal{C}) \\
\equiv & \text{“Predicate laws A.18, A.1”} \\
 & \neg(\mathcal{A} \wedge \mathcal{B}) \vee \mathcal{C} \\
\equiv & \text{“Predicate law A.22”} \\
 & \mathcal{A} \wedge \mathcal{B} \Rightarrow \mathcal{C} \ .
\end{array}
$$

Note the use of equivalence to replace a *part* of a formula, leading to an equivalence for the *whole* formula. That is the usual rule in mathematics: we can substitute equals for equals. But some of our laws are entailments $\Rrightarrow$, not equivalences; their substitution within formulae leads either to overall entailment or to its converse $\Lleftarrow$. Entailment distributes through quantification, conjunction, disjunction, and the consequent of implication; it is reversed in negations and antecedents of implications. It does not distribute at all through equivalence $\Leftrightarrow$.

Here is an example of distribution. Suppose we have $\mathcal{A} \Rrightarrow \mathcal{A}'$, $\mathcal{B} \Lleftarrow \mathcal{B}'$, and $\mathcal{C} \equiv \mathcal{C}'$. Then we can proceed as follows:

$$
\begin{array}{ll}
 & (\mathcal{A} \Rightarrow \mathcal{B}) \Rightarrow \mathcal{C} \\
\Rrightarrow & \text{“since } \mathcal{A} \Rrightarrow \mathcal{A}' \text{”} \\
 & (\mathcal{A}' \Rightarrow \mathcal{B}) \Rightarrow \mathcal{C} \\
\Rrightarrow & \text{“since } \mathcal{B} \Lleftarrow \mathcal{B}' \text{”} \\
 & (\mathcal{A}' \Rightarrow \mathcal{B}') \Rightarrow \mathcal{C} \\
\equiv & \text{“since } \mathcal{C} \equiv \mathcal{C}' \text{”} \\
 & (\mathcal{A}' \Rightarrow \mathcal{B}') \Rightarrow \mathcal{C}' \ .
\end{array}
$$

2.9 Exercises

Ex. 2.1 Which of these are terms?

1. true
2. 17
3. $\mathsf{log}^2\, x$
4. $\mathsf{log}\,\mathsf{log}\, x$
5. $(\mathsf{log}\, x)^2$
6. $\mathsf{log}\, x^2$
7. $2x$
8. $x < x + 1$

Ex. 2.2 Write terms for the following:

1. The square root of the factorial of n.
2. The factorial of the square root of n.

Ex. 2.3 ♡ Which of these are propositional formulae?

1. true
2. $true$
3. true
4. $x < y \Rightarrow z$
5. $x < y \Rrightarrow z$
6. $x < y \Rrightarrow y < z$
7. $x < y \Rightarrow y < z$

Ex. 2.4 Assuming that all variables denote natural numbers, which of these propositional formulae are true in all states?

1. $x \geq 0$
2. $x < y \Rightarrow x + 1 \leq y$
3. $x \leq y \vee y \leq x$
4. $x \leq y \wedge y \leq x \Rightarrow x = y$
5. $x < y \wedge y < x \Rightarrow x = y$

6. $x < y \wedge y < x \Rightarrow x \neq y$
7. $x < y \vee y < x \Rightarrow x \neq y$

Ex. 2.5 ♡ Assuming that the one-place predicates even, odd mean 'is an even number', 'is an odd number' respectively, write general formulae for the following:

1. Every integer is either even or odd.
2. Every odd natural number is one more than some even natural number.
3. There is an even integer that is not one more than any odd natural number.
4. Zero is the least natural number.
5. There is no least integer.
6. Given any positive real number, there is another real number strictly between it and zero.

Ex. 2.6 ♡ Recall that $(\exists\, x \bullet \mathcal{A})$ means 'there is *at least* one x such that $\mathcal{A}$'. Write another formula that means 'there is *at most* one x such that $\mathcal{A}$'.

Ex. 2.7 (Recall Exercise 2.6.) Write a formula that means 'there is *exactly* one x such that $\mathcal{A}$'.

Ex. 2.8 ♡ Use the truth tables of Figure 2.3 to show that these formulae are true in all states:

1. $\mathcal{A} \Rightarrow (\mathcal{B} \Rightarrow \mathcal{A})$
2. $(\mathcal{A} \Rightarrow (\mathcal{B} \Rightarrow \mathcal{C})) \Rightarrow ((\mathcal{A} \Rightarrow \mathcal{B}) \Rightarrow (\mathcal{A} \Rightarrow \mathcal{C}))$
3. $(\neg\mathcal{A} \Rightarrow \neg\mathcal{B}) \Rightarrow (\mathcal{B} \Rightarrow \mathcal{A})$

Ex. 2.9 ♡ Show that $\mathcal{A} \Rrightarrow \mathcal{B} \Rightarrow \mathcal{A}$. *Hint*: Recall Exercise 2.8, and the meaning of $\Rrightarrow$.

Ex. 2.10 ♡ Prove this, using laws from Appendix A:

$$(\exists\, x \bullet (\mathcal{A} \Rightarrow \mathcal{B}) \wedge (\neg\mathcal{A} \Rightarrow \mathcal{C})) \equiv (\exists\, x \bullet \mathcal{A} \wedge \mathcal{B}) \vee (\exists\, x \bullet \neg\mathcal{A} \wedge \mathcal{C})\,.$$

Ex. 2.11 Suppose $\mathcal{N}$ contains no free x. Prove this:

$$(\exists\, x \bullet (\mathcal{N} \Rightarrow \mathcal{A}) \wedge (\neg\mathcal{N} \Rightarrow \mathcal{B})) \equiv (\mathcal{N} \Rightarrow (\exists\, x \bullet \mathcal{A})) \wedge (\neg\mathcal{N} \Rightarrow (\exists\, x \bullet \mathcal{B}))\ .$$

Hint: Recall Exercise 2.10.

Ex. 2.12 ♡ Prove this, for any formula $\mathcal{A}$:

$$(\exists\, a \bullet (\forall\, b \bullet \mathcal{A})) \quad \Rrightarrow \quad (\forall\, b \bullet (\exists\, a \bullet \mathcal{A}))\,.$$

Is the converse true?

Ex. 2.13 Show that $(\exists\, x, y \bullet x \neq y) \equiv (\forall\, x \bullet (\exists\, y \bullet x \neq y))$. *Hint*: To show $\mathcal{A} \equiv \mathcal{B}$, show $\mathcal{A} \Rrightarrow \mathcal{B} \Rrightarrow \mathcal{A}$.