II.1 SOFL logic

SOFL logic is an extension of classical propositional logic and predicate logic; it allows “undefined” as a logical value

(SOFL adopts the three-value logic used in VDM).

II.1.1 Propositional logic

Definition: A proposition is a statement that is either true or false.

For example, the following statements are propositions:

(1) Tiger is animal (true)

(2) Apple is fruit (true)

(3) $3 + 5 > 10$ (false)

In contrast, the following statements are not propositions:

(1) Are you happy?

(2) Let's go swimming

(3) x := y + 3 (assignment statement)

Definition: The value true and false are called truth value.

In SOFL we use bool to represent the boolean type that contains the truth values, that is:

bool = {true, false}

Propositions are represented by symbols:

(1) P: Tiger is animal.
(2) Q: Apple is fruit.
(3) R: $3 + 5 > 10$.

Such a proposition is called atomic proposition (which cannot be decomposed).

Propositions can be connected using logical operators to form propositional expressions (or compound propositions) that describe more complex propositions.

Propositional operators

operator	read as	priority
not	not	highest
and	and	
or	or	
=>	implies	
<=>	is equivalent to	lowest

Conjunction

Definition: A conjunction is a propositional expression whose principal operator is and.

For example:

x > 5 and x < 10

Question: How to decide the truth value of a conjunction?

Truth table for conjunction

P1	P2	P1 and P2
true	true	true
true	false	false
false	true	false
false	false	false

Examples:

true and true <=> true
false and true <=> false
false and false <=> false

Disjunction

Definition: A disjunction is a propositional expression whose principal operator is or.

P1 or P2

For example:

$x > 5$ or $x < 3$

P1	P2	P1 or P2
true	true	true
true	false	true
false	true	true
false	false	false

Negation

Definition: A negation is a propositional expression whose principal operator is not.

not P1

Example:

not x > 5

P1	not P1
true	false
false	true

Implication

Definition: An implication is a propositional expression whose principal operator is =>.

P1 => P2

P1	P2	P1 => P2
true	true	true
true	false	false
false	true	true
false	false	true

Example:

x > 10 => x > 5

In this case we can also say that x > 10 is stronger than x > 5.

Equivalence

Definition: An equivalence is a propositional expression whose principal operator is <=>.

P1 <=> P2

P1	P2	P1 <=> P2
true	true	true
true	false	false
false	true	false
false	false	true

Examples:

(1)John is Chris' friend <=> Chris is John's friend

(2) x > 10 <=> not x = 10 and not x < 10

The use of parentheses

An expression is interpreted by applying the operator priority order unless parentheses are used.

For example: the expression

not p and q or r <=> p => q and r

is equivalent to the expression:

(((not p) and q) or r) <=> (p => (q and r))

Parentheses can be used to change the precedence of operators in expressions. For example, the above expression can be changed to:

not (p and ((q or (r <=> p))=> q) and r)

Tautology, contradiction, and contingency

Definition: A tautology is a proposition that evaluates to true in every combination of the truth values of its constituent propositions.

Examples:

(1) P or not P

(2) x > 10 or x <= 10

Definition: A contradiction is a proposition that evaluates to false in every combination of the truth values of its constituent propositions.

In other words, a contradiction is a negation of a tautology.

Examples:

(1) P and not P

(2) $x > 10$ and $x < 10$

Definition: A contingency is a propositional expression that is neither a tautology nor a contradiction.

In other words, a contingency can evaluate to either true or false.

Examples:

(1) P and Q (P and Q are not related with each other)

(2) $x > 5$ or $x < -5$

Normal forms

Definition: A disjunctive normal form is a special kind of disjunction in which each constituent propositional expression, is a conjunction of atomic propositions or their negations.

Form:

P_1 or P_2 or ... or P_n

and

P_i = Q_i_1 and Q_i_2 and … and Q_i_m

where i = 1,…,n and Q_i_j (j = 1, …., m) is an atomic proposition or negation of an atomic proposition.

The characteristic of a disjunctive normal form:

It evaluates to true as long as one of the constituent expression evaluates to true.

II.1.2 Predicate logic

The propositional logic only allows to make statements about specific objects, but it does not allow us to make universal statements and existential statements.

For example, the following are universal statements:

(1) Every student of Hiroshima University is happy.

(2) Nobody knows what to happen tomorrow.

The following are existential statements:

(1) One of my classmates received an award.

(2) Some students in my class do not like mathematics.

Predicates

Definition: A predicate is a truth-valued function.

In other words, a predicate is a function from a set X to the boolean type bool:

$$p: X \rightarrow bool$$

For example:

$x > 10$ is a predicate, but not proposition.

$5 > 10$ is a proposition, derived from the predicate $x > 10$ by substituting 5 for x.

where x is an integer variable.

Basic types (sets) in SOFL

The following are basic types in SOFL:

nat0: 0, 1, 2, 3, 4, … (natural numbers including 0)

nat: 1, 2, 3, 4, 5, … (natural numbers)

int: … -2, -1, 0, 1, 2, …(integers)

real: …-2.5, -1.4, 0, 1.4, 2.5,(real numbers)

char: ‘a’, ‘b’, ‘x’, ‘%’, … (characters)

bool: true, false (boolean values)

Example: a predicate ``is_big” is defined as follows:

is_big(x: int): bool
== x > 10

Then, the following propositions can be formed:

is_big(10) (false)
is_big(15) (true)
is_big(9) (false)

Quantifiers

(1) Universal quantifier

For example:
is-big(x) == x > 10 (is-big is a predicate)

Then we can write the conjunction

is-big(12) and is-big(15) and is-big(20)

as

forall[x: {12, 15, 20}] | is-big(x)

In general, the universally quantified expression has the form:

forall[x1: X1, …, xn: Xn] | p(x1, x2, …, xn)

forall --- universal quantifier

xi: Xi (i =1,...,n) --- bindings

x1, x2, …, xn --- bound variables

X1, X2, …, Xn --- the ranges (sets or types) of the bound variables

p(x1, x2, …, xn) --- predicate

(2) Existential quantifier

For example, we can write the disjunction

is-big(5) or is-big(12) or is-big(15)

as

exists[x: {5, 12, 15}] | is-big(x)

We call such an expression existentially quantified expression.

exists![x: T] | p(x)

means that there exists a unique x in T that satisfies condition p(x).

In general, the existentially quantified expression has the form:

exists[x1: X1, …, xn: Xn] | p(x1, x2, …, xn)

exists --- existential quantifier

xi: Xi (i=1,...,n) --- bindings

x1, x2, …, xn --- bound variables

X1, X2, …, Xn --- the ranges (sets or types) of the bound variables

p(x1, x2, …, xn) --- predicate

(3) The convention

The body of a quantified expression is considered to extend as far as the right possible.

Example: the quantified expression

forall[x: nat] | (x > z and (exists[y: nat] | y > x))

is equivalent to the expression:

forall[x: nat] | x > z and exists[y: nat] | y > x

Multiple quantifiers

Examples:

forall[x: X] | forall[y: Y] | p(x,y)

can be written as:

forall[x: X, y: Y] | p(x, y)

forall[x: X] | exists[y: Y] | p(x, y)

can be written as:

forall[x: X]exists[x: Y] | p(x, y)

----

exists[x: X] | exists[y: Y] | p(x, y)

can be written as:

exists[x: X, y: Y] | p(x, y)

Examples of multiple quantifiers

forall[i: nat] exists[j: nat] | j > i

This predicate is true, but the inversion of the universal quantifier and the existential quantifier will change the truth of the expression. Consider the following quantified expression, which is not true.

exists[j: nat] forall[i: nat] | j > i

Treatment of partial predicates

Definition: If a predicate may not yield a truth value for some values bound to its free variables, we call the predicate partial predicate.

For example,

$$x / 0 > 5$$

is a partial predicate.

The problem is that predicate logic does not allow undefined “value” to join evaluation of predicates.

One way to deal with this problem is to extend the truth tables of all the logical operators (i.e., and, or, not, =>, <=>) to allow the special value "undefined" to participate in evaluations of predicates. We use nil to denote "undefined".

The extension is made in the way that a result is given whenever possible, according to the predicate logic.

(and)	true	nil	false
true	true	nil	false
nil	nil	nil	false
false	false	false	false

(or)	true	nil	false
true	true	true	true
nil	true	nil	nil
false	true	nil	false

(not)	
true	false
nil	nil
false	true

Examples:

nil and false <=> false

nil and true <=> nil

nil and nil <=> nil

(=>)	true	nil	false
true	true	nil	false
nil	true	nil	nil
false	true	true	true

(<=>)	true	nil	false
true	true	nil	false
nil	nil	nil	nil
false	false	nil	true

Class exercise 1

Use predicate expressions to describe the following statements:

(1) Every integer is greater than 0, equal to 0, or less than 0.

(2) For any three real numbers *a*, *b*, and *c*, if *a* is greater than *b* and *b* is greater than *c*, then a will be greater than *c*.

(3) For any natural number *a* there must exist another natural number *b* such that *b* is greater than *a*.