Fondamenti della Programmazione: Metodi Evoluti

Prof. Enrico Nardelli

Lezione 5: Logica

Reminder: contracts

Associated with an individual feature:

- Pre-conditions (must be true BEFORE feature execution)
- Post-conditions (must be true AFTER feature execution)

Associated with a class:

- Class invariants (expresses consistency requirements between queries of a class)

Logic

How to express conditions in contracts?

We need a mathematical notation since conditions have to be automatically checked

Logic is the answer!

Reasoning and programming

Logic is the basis of

- Mathematics: proofs are only valid if they follow the rules of logic.

- Software development:
  - Conditions in program actions: "If *m* is positive, then execute this instruction"

  - Conditions in contracts:

    "x must not be zero, so that we can calculate $\frac{x+7}{x}$ "

Boolean expressions

A condition is expressed as a boolean expression.

It consists of

- Boolean variables (identifiers denoting boolean values)
- Boolean operators (**not**, **and**, **or**, **implies**, **=**)

and represents possible

- Boolean values (truth values, either **True** or **False**)

Examples

Examples of boolean expressions
(with *rain_today* and *cuckoo_sang_last_night* as boolean variables):

- *rain_today*
  (a boolean variable is a boolean expression)
- **not** *rain_today*
- (**not** *cuckoo_sang_last_night*) **implies** *rain_today*

(Parentheses group sub-expressions)

Negation (not)

a	**not** *a*
True	**False**
False	**True**

For any boolean expression *e* and any values of variables:

- Exactly one of *e* and **not** *e* has value **True**
- Exactly one of *e* and **not** *e* has value **False**
- One of *e* and **not** *e* has value **True** (Principle of the Excluded Middle)
- Not both of *e* and **not** *e* have value **True** (Principle of Non-Contradiction)

Conjunction (and)

a	*b*	*a* **and** *b*
True	**True**	**True**
True	**False**	**False**
False	**True**	**False**
False	**False**	**False**

and operator is commutative

and operator is associative

- *a* **and** (*b* **and** *c*) = (*a* **and** *b*) **and** *c*

Conjunction principle:

- An **and** conjunction has value **False** except if both operands have value **True**

Disjunction (or)

a	*b*	*a* **or** *b*
True	**True**	**True**
True	**False**	**True**
False	**True**	**True**
False	**False**	**False**

or operator is commutative

or operator is associative:

- *a* **or** (*b* **or** *c*) = (*a* **or** *b*) **or** *c*

Disjunction principle:

- An **or** disjunction has value **True** except if both operands have value **False**

NB: differently from 'or' in common language **or** is non-exclusive

Truth assignment and truth table

Truth assignment for a set of variables: particular choice of values (**True** or **False**), for every variable

A truth assignment satisfies an expression if the value for the expression is **True**

A truth table for an expression with n variables has

- $n + 1$ columns
- 2^n rows

Combined truth table for basic operators

a	*b*	**not** *a*	*a* **or** *b*	*a* **and** *b*
True	True	False	True	True
True	False		True	False
False	True	True	True	False
False	False		False	False

Tautologies

Tautology: a boolean expression that has value **True** for every possible truth assignment

Examples:

- *a* **or** (**not** *a*)
- **not** (*a* **and** (**not** *a*))
- (*a* **and** *b*) **or** ((**not** *a*) **or** (**not** *b*))

Contradictions

Contradiction: a boolean expression that has value **False** for every possible truth assignment

Examples:

- *a* **and** (**not** *a*)
- **not** (*a* **or** (**not** *a*))

Satisfiable: for at least one truth assignment the expression yields **True**

- Any tautology is satisfiable
- No contradiction is satisfiable.

Equivalence (=)

a	b	$a = b$
True	**True**	**True**
True	**False**	**False**
False	**True**	**False**
False	**False**	**True**

= operator is commutative ($a = b$ has same value as $b = a$)

= operator is reflexive ($a = a$ is a tautology for any a)

Substitution:

- For any expressions u, v and any expression e containing u, if $u = v$ is a tautology and e' is the expression obtained from e by replacing every occurrence of u by v, then $e = e'$ is a tautology

Types of propositions (boolean expressions)

Tautology

- **True** for all truth assignments
  - P **or** (**not** P)
  - **not** (P **and** (**not** P))
  - (P **and** Q) **or** ((**not** P) **or** (**not** Q))

Contradiction

- **False** for all truth assignments
  - P **and** (**not** P)

Satisfiable

- **True** for at least one truth assignment

Equivalent

- φ and χ are equivalent if they are satisfied under exactly the same truth assignments, or if φ = χ is a tautology

De Morgan's laws

They show the duality between **and** and **or**: negating an expression is equivalent to negating variables and swapping **and** and **or**

They are tautologies

- **not** (*a* **or** *b*) = (**not** *a*) **and** (**not** *b*)
- **not** (*a* **and** *b*) = (**not** *a*) **or** (**not** *b*)

- *a* **or** *b* = **not** ((**not** *a*) **and** (**not** *b*))
- *a* **and** *b* = **not** ((**not** *a*) **or** (**not** *b*))

More tautologies (distributivity):

- (*a* **and** (*b* **or** *c*)) = ((*a* **and** *b*) **or** (*a* **and** *c*))
- (*a* **or** (*b* **and** *c*)) = ((*a* **or** *b*) **and** (*a* **or** *c*))

Syntax convention: binding of operators

Order of binding (starting with tightest binding) or precedence among operators:

not, **and**, **or**, **implies**, **=**

Style rules:

When writing a boolean expression, drop the parentheses:

- Around the expressions of each side of "=" if whole expression is an equivalence.
- Around successive elementary terms if they are separated by the same associative operators.

Implication (implies)

a	*b*	*a* **implies** *b*
True	**True**	**True**
True	**False**	**False**
False	**True**	**True**
False	**False**	**True**

a **implies** *b*, for any *a* and *b*, is the value of (**not** *a*) **or** *b*

In *a* **implies** *b*: *a* is antecedent, *b* consequent

Implication principle:

- An implication has value **True** except if its antecedent has value **True** and its consequent has value **False**
- In particular, always **True** if antecedent is **False**

Implication in ordinary language

implies in ordinary language often means **causation**, as in "if … then …"

- *"If the weather stays like this, skiing will be great this week-end"*
- *"If you put this stuff in your hand luggage, they won't let you through."*

Misunderstanding implication

Whenever *a* is **False**,

the expression "*a* **implies** *b*" is **True**, regardless of *b*

- "Today is Sunday *implies* 2+2=5."
- "2+2=5 *implies* today is Sunday."

Both of the above expressions are **True**

Cases in which *a* is **False** tell us nothing about the truth of the consequent

Reversing implications (1)

It is not generally true that

~~*a* **implies** *b* = (**not** *a*) **implies** (**not** *b*)~~

Example (wrong!):

- *"All the people in Rome who live near Spanish Steps are rich. I do not live near Spanish Steps, so I am not rich."*

live_near_spanish_steps **implies** *rich* [1]

~~(**not** *live_near_spanish_steps*) **implies** (**not** *rich*)~~ [2]

Reversing implications (2)

Correct:

a **implies** *b* = (**not** *b*) **implies** (**not** *a*)

Example:

- "*All the people who live near Spanish Steps are rich. She is not rich, so she can't be living near Spanish Steps*"

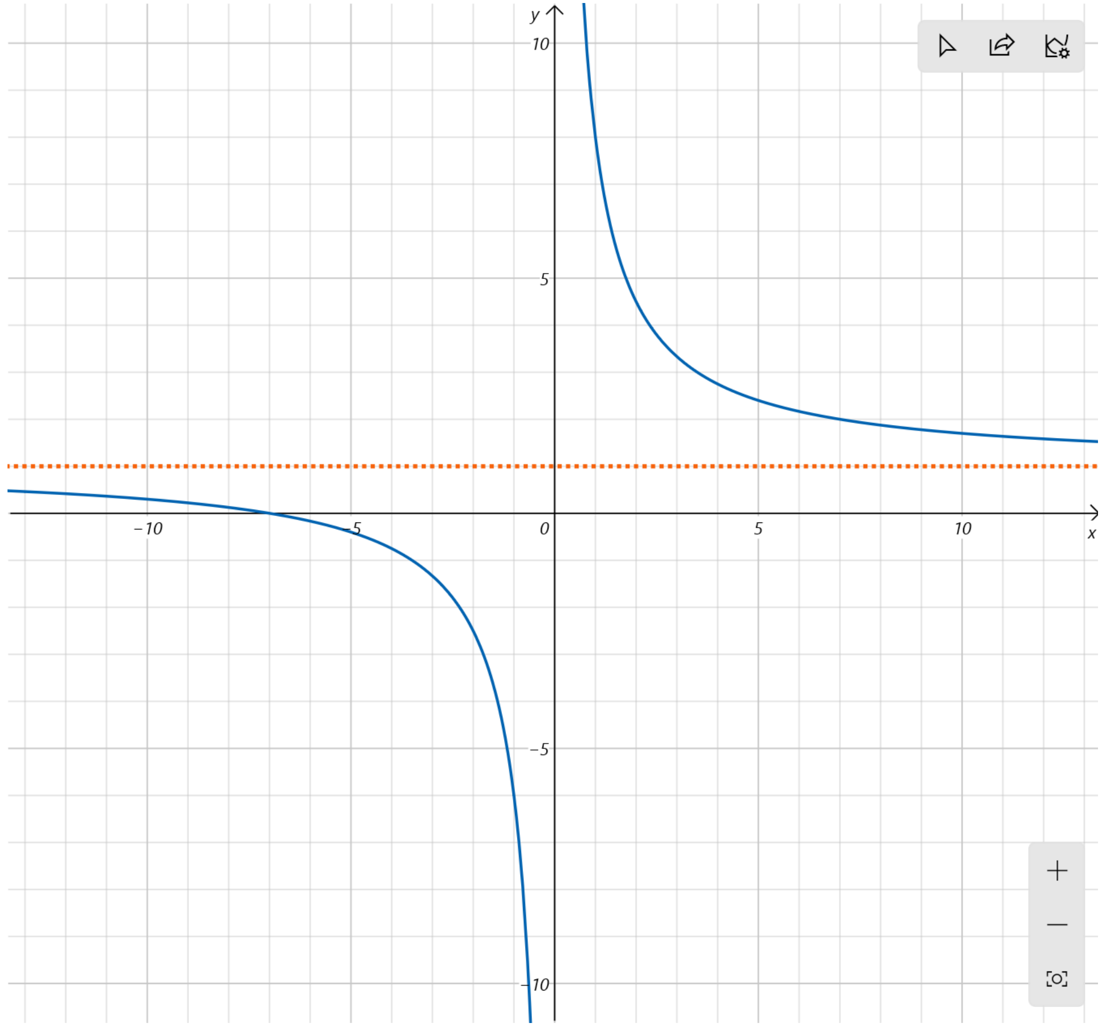
live_near_spanish_steps **implies** *rich* =

(**not** *rich*) **implies** (**not** *live_near_spanish_steps*)

Semistrict boolean operators (1)

Example boolean-valued expression (x is an integer):

$$\frac{x + 7}{x} > 0$$



True for $x < -7$ or $x > 0$

False for $x \geq -7$ and $x < 0$

Undefined for $x = 0$

Semistrict boolean operators (2)

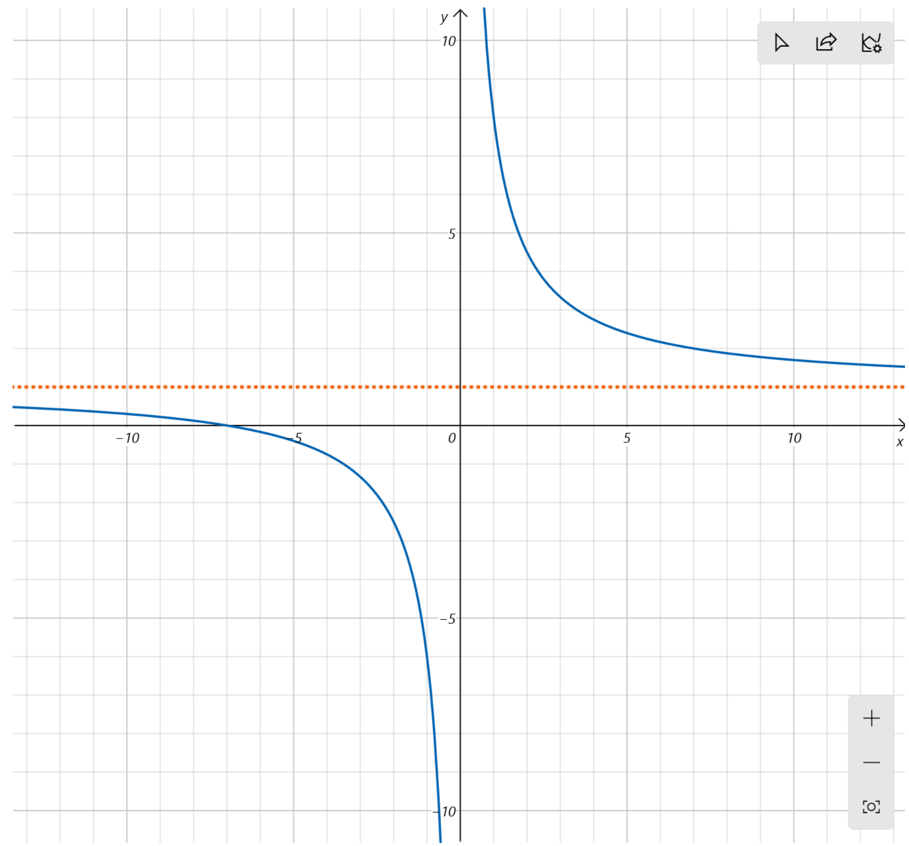
- Avoid division by zero through a logic condition:

$(x \mathrel{/=} 0)$ **and** $(((x + 7) \,/\, x) > 0)$

True for $x < -7$ or $x > 0$

False for $x \geq -7$ and $x \leq 0$

Is always defined !



Semistrict boolean operators (3)

BUT:

- What happens when evaluating it?
- Program could crash during evaluation of

  $(x \mathrel{/=} 0)$ **and** $(((x + 7) \mathbin{/} x) > 0)$

We need a non-commutative version of **and** (and **or**):

Semistrict boolean operators

Semistrict operators (and then, or else)

a **and then** *b*: has same value as *a* **and** *b* if *a* and *b* are both defined, and has **False** whenever *a* has value **False** even if b is undefined

a **or else** *b*: has same value as *a* **or** *b* if *a* and *b* are both defined, and has **True** whenever *a* has value **True** even if b is undefined

$(x \mathrel{/=} 0)$ **and then** $(((x + 7) \mathbin{/} x) > 0)$

Semistrict operators allow us to define an order of expression evaluation (left to right).

Important for programming when undefined objects may cause program crashes

Ordinary vs. Semistrict boolean operators

Use

- Ordinary boolean operators (**and** and **or**) if you can guarantee that both operands are defined
- **and then** if an additional condition only makes sense when the previous one(s) are true
- **or else** if an additional condition only makes sense when the previous one(s) are false

Example:

- "If you are not single, then your spouse must sign the contract"

not *is_single* **and then** *spouse_must_sign*

is_single **or else** *spouse_must_sign*

Semistrict implication

Example with implies:

- "If you are not single, then your spouse must sign the contract."

(**not** *is_single*) **implies** *spouse_must_sign*

Definition of **implies**: in our case, always semistrict!

- *a* **implies** *b* = (**not** *a*) **or else** *b*

Strict or semi-strict?

Hands-On

- *a* = 0 **or** *b* = 0
- *a* /= 0 **and** *b* // *a* /= 0
- *a* /= **Void and** *b* /= **Void**
- *a* < 0 **or** *sqrt* (*a*) > 2
- (*a* = *b* **and** *b* /= **Void**) **and** *a.age* = 0

Programming language notation for boolean operators

Eiffel keyword	Common mathematical symbol
not	~ or ¬
or	$\vee$
and	$\wedge$
=	$\Leftrightarrow$
implies	$\Rightarrow$

Propositional and predicate calculus

Propositional calculus:

property *p* holds for a single object

Predicate calculus:

property *p* holds for several objects

Generalizing or

G : group of objects, *p* : property

Generalization of **or**:

Does *at least one* of the objects in *G* satisfy *p* ?

Is at least one station of Line 8 an exchange?

Station_Balard.is_exchange **or** *Station_Lourmel.is_exchange* **or** *Station_Boucicaut.is_exchange* **or**

... (all stations of Line 8)

Existential quantifier: *exists*, or ∃

∃ *s* : *Stations_8* | *s.is_exchange*

"There exists an *s* in *Stations_8* such that *s.is_exchange* is true"

Generalizing and

Generalization of **and**:

Does *every* object in *G* satisfy p?

Are all stations of Tram 8 exchanges?

Station_Balard.is_exchange **and** *Station_Lourmel.is_exchange* **and** *Station_Boucicaut.is_exchange* **and** …

(all stations of Line 8)

Universal quantifier: *for_all*, or ∀

∀ *s*: *Stations_8* | *s.is_exchange*

"For all *s* in *Stations8* | *s.is_exchange* is true"

Existentially quantified expression

Boolean expression:

$\exists$ *s* : *SOME_SET* | *s.some_property*

- *True* if and only if at least one member of *SOME_SET* satisfies property *some_property*

Proving

- **True**: Find one element of *SOME_SET* that satisfies the property
- **False**: Prove that no element of *SOME_SET* satisfies the property (test all elements)

Universally quantified expression

Boolean expression:

$$\forall\ s: SOME\_SET \mid s.some\_property$$

- *True* if and only if every member of *SOME_SET* satisfies property *some_property*

Proving

- **True**: Prove that every element of *SOME_SET* satisfies the property (test all elements)
- **False**: Find one element of *SOME_SET* that does not satisfies the property

Duality

Generalization of DeMorgan's laws:

$$\textbf{not}\ (\exists\ s : SOME\_SET \mid P) = \forall\ s : SOME\_SET \mid \textbf{not}\ P$$

$$\textbf{not}\ (\forall\ s : SOME\_SET \mid P) = \exists\ s : SOME\_SET \mid \textbf{not}\ P$$

Empty sets

- ∃ s : SOME_SET | *some_property*

  If SOME_SET is empty: always **False**

(there is no element in SOME_SET therefore there is none able to satisfy *some_property*)

Then, by duality we have

- ∀ s : SOME_SET | *some_property*

  If SOME_SET is empty: always **True**

Tautology / contradiction / satisfiable?

Hands-On

Let the range of variables be INTEGER

x < 0 **or** x >= 0

x > 0 **implies** x > 1

∀x | x > 0 **implies** x > 1

∀x | x∗y = y

∃y | ∀x | x∗y = y