Einführung in die Programmierung
Introduction to Programming

Prof. Dr. Bertrand Meyer

Lecture 5: Invariants and Logic
Programming is reasoning.
Logic is the science of reasoning.
We use logic in our every days life:

“Socrates is human.
All humans are mortal.

Therefore Socrates must be mortal.”
Reasoning and programming

Logic is the basis of

- **Mathematics**: proofs are only valid if they follow the rules of logic.
- **Software development**:  
  - Conditions in contracts:  
    “$x$ must not be zero, so that we can calculate $\frac{x+7}{x}$.”
  - Conditions in program actions: “If $i$ is positive, then execute this instruction.” (to be introduced in a later lecture)
Boolean expressions

A condition is expressed as a boolean expression. It consists of

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

and represents possible

- boolean values (truth values, either True or False).
Examples of boolean expressions
(with \textit{rain\_today} and \textit{cuckoo\_sang\_last\_night} as boolean variables):

- \textit{rain\_today}
  (a boolean variable is a boolean expression)
- \texttt{not} \textit{rain\_today}
- \texttt{(not cuckoo\_sang\_last\_night)} implies \textit{rain\_today}

(Parentheses group sub-expressions.)
Negation (\textbf{not})

<table>
<thead>
<tr>
<th>$a$</th>
<th>not $a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

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.)
**Disjunction (or)**

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$a$ or $b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

*or* operator is **non-exclusive**.

*or* operator is **commutative**.

**Disjunction principle:**

- An *or* disjunction has value **True** except if both operands have value **False**.
Conjunction (**and**) is **commutative**.

**Duality of and and or:** properties of either operator yield properties of other (negating + swapping True and False)

**Conjunction principle:**

- An and conjunction has value False except if both operands have value True.
Complex expressions

Build more complex boolean expressions by using the boolean operators.

Example:

\[ a \text{ and } (b \text{ and } \neg c) \]
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

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>not $a$</th>
<th>$a$ or $b$</th>
<th>$a$ and $b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td></td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td></td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>
Tautologies

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

Examples:

- $a \text{ or } \neg a$
- $\neg (a \text{ and } \neg a)$
- $(a \text{ and } b) \text{ or } ((\neg a) \text{ or } (\neg b))$
Contradictions

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

Examples:

- \( a \text{ and } (\text{not } a) \)

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

- Any tautology is satisfiable.
- No contradiction is satisfiable.
# Equivalence (=)

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$a = b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

- $=$ 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 $e$, 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.
De Morgan’s laws

De Morgan’s Laws: Tautologies

- \((\neg (a \lor b)) = ((\neg a) \land (\neg b))\)
- \((\neg (a \land b)) = ((\neg a) \lor (\neg b))\)

More tautologies:

- \((a \land (b \lor c)) = ((a \land b) \lor (a \land c))\)
- \((a \lor (b \land c)) = ((a \lor b) \land (a \lor c))\)
Binding

Order of binding (starting with tightest binding): **not**, **and**, **or**, **implies** (to be introduced), =.

**and** and **or** are **associative**:

- $a \text{ and } (b \text{ and } c) = (a \text{ and } b) \text{ and } c$
- $a \text{ or } (b \text{ or } c) = (a \text{ or } b) \text{ or } c$

**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 (\textit{implies})

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$a$ implies $b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

\textit{a implies b}, for any \textit{a} and \textit{b}, is the value of (\textit{not a}) \textit{or} \textit{b}

In \textit{a implies b}: \textit{a} is antecedent, \textit{b} consequent

**Implication principle:**

- An implication has value \textit{True} except if its antecedent has value \textit{True} and its consequent has value \textit{False}
- In particular, always \textit{True} if antecedent is \textit{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 implications

Whenever $a$ is $False$, $a$ implies $b$ is $True$, regardless of $b$:

- “If today is Wednesday, $2+2=5$.”
- “If $2+2=5$, today is Wednesday.”

Both of the above implications 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

\[ a \implies b = (\neg a) \implies (\neg b) \]

Example (wrong!):

- “All the people in Zurich who live near the lake are rich. I do not live near the lake, so I am not rich.”

\[ \text{live\_near\_lake} \implies \text{rich} \quad [1] \]

\[ (\neg \text{live\_near\_lake}) \implies (\neg \text{rich}) \quad [2] \]
Reversing implications (2)

Correct:

\[ a \implies b = (\neg b) \implies (\neg a) \]

Example:

- “All the people who live near the lake are rich. She is not rich, so she can’t be living in Küsnacht”

\[ \text{live\_near\_lake \implies rich} = (\neg \text{rich}) \implies (\neg \text{live\_near\_lake}) \]
Implication

VERNÜFTIGE FAHRZEUGLENKER PARKIEREN NICHT HIER!
FÜR DIE ANDEREN IST ES VERBOTEN!

AUS ARCHITEKTEN UND PLANER 31.5.1989
Semistrict boolean operators (1)

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

\[
\begin{array}{c}
x + 7 \\
x \\
\end{array} > 1
\]

False for \(x \leq -7\)
Undefined for \(x = 0\)
Semistrict boolean operators (2)

BUT:

- **Division by zero:** \( x \) must not be 0.

\[(x \neq 0) \text{ and } \left( \frac{x+7}{x} > 0 \right)\]

False for \( x \leq -7 \)
False for \( x = 0 \)
Semistrict boolean operators (3)

BUT:

- program would crash during evaluation of division

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

Non-strict boolean operators.
Non-strict operators (\textbf{and then, or else})

\textit{a and then b}: has same value as \textit{a and b} if \textit{a} and \textit{b} are defined, and has \textit{False} whenever \textit{a} has value \textit{False}.

\textit{a or else b}: has same value as \textit{a or b} if \textit{a} and \textit{b} are defined, and has \textit{True} whenever \textit{a} has value \textit{True}.

\((x \neq 0) \text{ and then } (\frac{x+7}{x} > 0)\)

Non-strict operators allow us to define an \textbf{order of expression evaluation} (left to right).

Important for programming when undefined objects may cause program crashes.
Ordinary vs. non-strict boolean operators

Use

- Ordinary boolean operators (\texttt{and} and \texttt{or}) if you can guarantee that both operands are defined.
- \texttt{and then}, if a condition only makes sense when another is true.
- \texttt{or else}, if a condition only makes sense when another is false.

Example:

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

\texttt{is\_single or else spouse\_must\_sign}
Non-strict implication

Example:
- “If you are not single, then your spouse must sign the contract.”

\[
\text{not is\_single} \implies \text{spouse\_must\_sign}
\]

Definition of \textit{implies}: in our case, \textit{always non-strict}!
- \( a \implies b = (\text{not } a) \text{ or else } b \)
# Eiffel keywords and mathematical symbols

<table>
<thead>
<tr>
<th>Eiffel keyword</th>
<th>Common mathematical symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>not</td>
<td>~ or ¬</td>
</tr>
<tr>
<td>or</td>
<td>∨</td>
</tr>
<tr>
<td>and</td>
<td>∧</td>
</tr>
<tr>
<td>=</td>
<td>⇔</td>
</tr>
<tr>
<td>implies</td>
<td>⇒</td>
</tr>
</tbody>
</table>
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

\textbf{or:} Does at least one of the objects in \(G\) satisfy \(p\)?

Is at least one station of Line 8 an exchange?

\(\text{Station\_Balard}.\text{is\_exchange or} \text{Station\_Lourmel}.\text{is\_exchange or} \text{Station\_Boucicaut}.\text{is\_exchange or} \ldots \) (all stations of Line 8)

Existential quantifier: \textit{exists}, or \(\exists\)

\(\exists \; s: \text{Stations\_8 \mid s.is\_exchange}\)

"There exists an \(s\) in \text{Stations\_8} such that \(s.is\_exchange\) is true"
Generalizing **and**

**and:** Does every object in $G$ satisfy $p$?
Are all stations of Tram 8 exchanges?

\[\text{Station\_Balard\_is\_exchange and Station\_Lourmel\_is\_exchange and Station\_Boucicaut\_is\_exchange and ...} \]
\[\text{(all stations of Line 8)}\]

**Universal quantifier:** *for all*, or $\forall$

\[\forall s: \text{Stations\_8 | s\_is\_exchange}\]

"For all $s$ in Stations8 | s\_is\_exchange is true"
Existentially quantified expression

Boolean expression:

\[ \exists s : \textit{SOME\_SET} \mid s.\textit{some\_property} \]

- *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:
\[ \forall s: \text{SOME}_{-}\text{SET} \mid s.\text{some}_\text{property} \]

- 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:

\[
\neg (\exists s : \text{SOME\_SET} \mid P) = \forall s : \text{SOME\_SET} \mid \neg P
\]

\[
\neg (\forall s : \text{SOME\_SET} \mid P) = \exists s : \text{SOME\_SET} \mid \neg P
\]
Empty sets

\[ \exists \ s : \text{SOME\_SET} \mid \text{some\_property} \]
\[ \text{with } \text{SOME\_SET} \text{ empty } \Rightarrow \text{always } \text{False} \]

\[ \forall \ s : \text{SOME\_SET} \mid \text{some\_property} \]
\[ \text{with } \text{SOME\_SET} \text{ empty } \Rightarrow \text{always } \text{True} \]
Key concepts

- Logic as a tool for reasoning
- Boolean operators: truth tables
- Properties of boolean operators: don’t use truth tables!
- Predicate calculus: to talk about logical properties of sets
- Non-strict boolean operators