Tableaux technique

Implementation of the tableaux technique by deduction in propositional logic

Cristian Andrei Mai Mihai - Mattia Tarantino

Sapienza Università di Roma

Logica e Informatica – A.A. 2022/23

What is the Tableau method?

The tableau method is a formal demonstration procedure, existing in many varieties and for different logics.

This is a refutation method:
Instead of proving directly that $\Gamma \models \phi$, we prove that $\Gamma \cup \{\neg\phi\}$ is an unsatisfiable set.

Theorem

$\Gamma \models \phi$ *se e solo se* $\Gamma \cup \{\neg\phi\}$ *è insoddisfacibile.*

Proof.

$\Rightarrow$ By absurdity, if $\Gamma \cup \{\neg\phi\}$ is satisfiable, then there exists a $\mathcal{M}$ model of $\Gamma \cup \{\neg\phi\}$: $\mathcal{M} \models \psi$ for each $\psi \in \Gamma$ and $\mathcal{M} \not\models \phi$.
Then $\Gamma \not\models \phi$
$\Leftarrow$ Absurdly, if $\Gamma \not\models \phi$, then there exists an interpretation $\mathcal{M}$ such that $\mathcal{M} \models \psi$ for every $\psi \in \Gamma$ and $\mathcal{M} \not\models \phi$.
So $\mathcal{M} \models \neg\phi$ and $\mathcal{M}$ is a model of $\Gamma \cup \{\neg\phi\}$:
$\Gamma \cup \{\neg\phi\}$ is satisfiable $\square$

Tableau construction

- This technique involves the construction of a binary tree, known as **tableau**, whose nodes are labeled by a set of formulas.
- The root of the tree consists of the initial set of formulas Γ and the negated conclusion $\neg\phi$.
- You apply the **expansion rules** for the logical connectives in the propositional formula, adding new nodes to the tableau tree.
- You continue to apply the expansion rules to the newly generated formulas until you can no longer do so, in which case the **saturation** of the tableau occurs.
- When the tableau is saturated and in each branch there is a formula and its negation (tableau **closed**), then $\Gamma \cup \{\neg\phi\}$ is unsatisfiable, so $\Gamma \models \phi$.
  Otherwise, we will have an **open** branch and then $\Gamma \not\models \phi$.

Expansion rules

The expansion rules are divided into:

- **Double negation**: Create a new child for each leaf in the tableau without the double negation
- **Alpha rules**: Create one new child for each leaf of the tableau based on the logical connective
- **Beta rules**: Create two new children for each leaf of the tableau based on the logical connective

α rules

$$\frac{\phi \wedge \psi}{\begin{array}{c}\phi \\ \psi\end{array}} \qquad \frac{\neg(\phi \vee \psi)}{\begin{array}{c}\neg\phi \\ \neg\psi\end{array}} \qquad \frac{\neg(\phi \to \psi)}{\begin{array}{c}\phi \\ \neg\psi\end{array}}$$

$\neg\neg$-Elimination

$$\frac{\neg\neg\phi}{\phi}$$

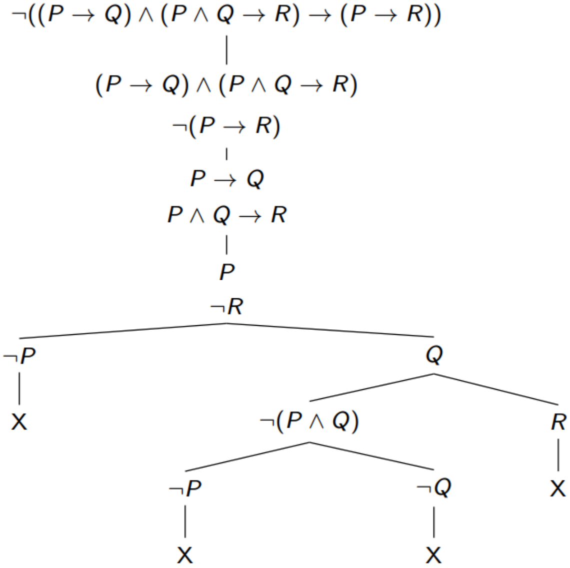
β rules

$$\frac{\phi \vee \psi}{\phi \mid \psi} \qquad \frac{\neg(\phi \wedge \psi)}{\neg\phi \mid \neg\psi} \qquad \frac{\phi \to \psi}{\neg\phi \mid \psi}$$

Branch Closure

$$\frac{\begin{array}{c}\phi \\ \neg\phi\end{array}}{X}$$

Example of tableau application



The implementation

- We treat formulas as binary trees whose leaves are associated with propositional variables, while intermediate nodes are associated with connectives.
- The class **Formula** represents the formula itself and is constructed from the root of the tree

```
class Formula:

    def __init__(self, root):
        # lista che ricorda i singoli termini all'interno della formula
        self.termini = []
        self.root = root
```

The implementation

The class **Node** represents the nodes of the formula where "value" is a connective or a variable, in case it is variable "boolean" represents its truth value.

```
class Node:
    def __init__(self, value):
        # eventuali figli
        self.children = []
        # valore che può essere un operatore o un termine
        self.value = value
        # valore di verità che è stato assegnato a un termine
        self.boolean = None

    # metodo per sapere se ho un termine
    def estTermine(self):
        return len(self.children) == 0

    # metodo per capire se un certo è un operatore
    def estOperator(self):
        operators = ["∧", "∨", "~", "→"]  # and, or, not, not not, implica
        if self.value in operators:
            return True
        else:
            return False
```

Example of formula representation as a tree:

$$\neg((p \to q) \land (p \land q \to r) \to (p \to r))$$

```
Rappresentazione della formula come albero:
~
  →
    ∧
      →
        p
        q
      →
        ∧
          p
          q
        r
    →
      p
      r
```

The implementation

The class **Tableaux** represents the tableau tree while **Node** its nodes.

```
class Tableaux():
    # classe che rappresenta i nodi dell'albero tableaux
    class Nodo():

        def __init__(self, ins_formula):
            # le formule che appartengono al singolo nodo (come albero)
            self.ins_formula = ins_formula
            self.children = []
            # le stesse formule ma in una lista per poter essere stampate (come stringa)
            self.formule = []
            for formula in ins_formula:
                self.formule.append(formula.to_string())

    def __init__(self, ins_formula):
        # valore che indica la soddisfacibilità dell'insieme di formule
        self.ret = False
        # una lista con tutti i termini nelle varie formule
        self.terms = []
        self.root = self.Nodo(ins_formula)
        for formula in ins_formula:
            for termine in formula.termini:
                if termine not in self.terms:
                    self.terms.append(termine)
        # numero di termini
        self.terms_len = len(self.terms)
```

The implementation

The **Resolve** method applies the tableau method starting from the root of the tree, in particular during its first operation it checks whether the **double negation** rule can be applied. If so, it creates a new tableau node for each leaf without the double negation.

```
def risolvi(self, nodo, foglie):
        # controllo per sapere quando fermarsi
        if self.ret == False:
            if nodo.ins_formula is None:
                return
            # inizio controllo dei nodi
            # 1° CONTROLLO: DOPPIA NEGAZIONE
            for formula in nodo.ins_formula:
                # controllo se c'è "~"
                if formula.root.value == "~":
                    # se si controllo i figli
                    # se è uno ed è "~"
                    if formula.root.children[0].value == "~":
                        # lista di supporto per sapere quali sono le foglie dell'albero tableaux
                        new_foglie = []
                        for node in foglie:
                            # crea una nuova formula che ha come root l'elemento dopo la doppia negazione
                            new_formula = Formula(formula.root.children[0].children[0])
                            # crea un nuovo nodo dell'albero tableaux
                            new_nodo = self.Nodo([new_formula])
                            node.children.append(new_nodo)
                            new_foglie.append(new_nodo)
                        foglie = new_foglie
```

The implementation

Next, a check for the applicability of the **Alpha rules** is performed:

$$\frac{\neg(\phi \vee \psi)}{\begin{array}{c}\neg\phi \\ \neg\psi\end{array}}$$

```
# 2° CONTROLLO: ALPHA RULE
for formula in nodo.ins_formula:
    # controllo se c'è "~"
    if formula.root.value == "~":
        # se il figlio è un "V" abbiamo ~(φ V ψ)
        if formula.root.children[0].value == "V":
            new_foglie = []
            for node in foglie:
                new_ins_formula = []
                # crea una nodo della formula ~φ
                phi = Node("~")
                phi.children.append(formula.root.children[0].children[0])
                new_ins_formula.append(Formula(phi))
                # crea una nodo della formula ~ψ
                psi = Node("~")
                psi.children.append(formula.root.children[0].children[1])
                new_ins_formula.append(Formula(psi))
                # crea un nodo di tableaux per ogni foglia e lo passa come figlio a ognuna
                new_nodo = self.Nodo(new_ins_formula)
                node.children.append(new_nodo)
                new_foglie.append(new_nodo)
            foglie = new_foglie
```

The implementation

$$\frac{\neg(\phi \rightarrow \psi)}{\begin{array}{c}\phi \\ \neg\psi\end{array}}$$

```
# se il figlio è un "→" abbiamo ~(φ → ψ)
if formula.root.children[0].value == "→":
    new_foglie = []
    for node in foglie:
        new_ins_formula = []
        # crea un nodo della formula φ
        phi = formula.root.children[0].children[0]
        new_ins_formula.append(Formula(phi))
        # crea una nodo della formula ~ψ
        psi = Node("~")
        psi.children.append(formula.root.children[0].children[1])
        new_ins_formula.append(Formula(psi))
        # crea un nodo di tableaux per ogni foglia e lo passa come figlio a ognuna
        new_nodo = self.Nodo(new_ins_formula)
        node.children.append(new_nodo)
        new_foglie.append(new_nodo)
    foglie = new_foglie
```

The implementation

$$\frac{\phi \wedge \psi}{\begin{array}{c}\phi\\ \psi\end{array}}$$

```
# controllo se c'è "∧" se si abbiamo φ ∧ ψ
if formula.root.value == "∧":
    new_foglie = []
    for node in foglie:
        new_ins_formula = []
        # crea un nodo della formula φ
        phi = formula.root.children[0]
        new_ins_formula.append(Formula(phi))
        # crea un nodo della formula ψ
        psi = formula.root.children[1]
        new_ins_formula.append(Formula(psi))
        # crea un nodo di tableaux per ogni foglia e lo passo come figlio a ognuna
        new_nodo = self.Nodo(new_ins_formula)
        node.children.append(new_nodo)
        new_foglie.append(new_nodo)
    foglie = new_foglie
```

The implementation

Next, a check for the applicability of the **Beta rules** is performed:

$$\frac{\neg(\phi \wedge \psi)}{\neg\phi \mid \neg\psi}$$

```
# 3° CONTROLLO: BETA RULE
for formula in nodo.ins_formula:
    # controllo se c'è "~"
    if formula.root.value == "~":
        # se il figlio è un "∧" abbiamo ~(φ ∧ ψ)
        if formula.root.children[0].value == "∧":
            new_foglie = []
            for node in foglie:
                ins_formula_sx = []
                ins_formula_dx = []
                # crea un nodo della formula ~φ
                phi = Node("~")
                phi.children.append(formula.root.children[0].children[0])
                ins_formula_sx.append(Formula(phi))
                # crea un nodo della formula ~ψ
                psi = Node("~")
                psi.children.append(formula.root.children[0].children[1])
                ins_formula_dx.append(Formula(psi))
                # crea un nodo del tableaux che ha come formula ~φ
                new_nodo_sx = self.Nodo(ins_formula_sx)
                # crea un nodo del tableaux che ha come formula ~ψ
                new_nodo_dx = self.Nodo(ins_formula_dx)
                node.children.append(new_nodo_sx)
                node.children.append(new_nodo_dx)
                new_foglie.append(new_nodo_sx)
                new_foglie.append(new_nodo_dx)
            foglie = new_foglie
```

The implementation

$$\frac{\phi \lor \psi}{\phi \mid \psi}$$

```python
# controllo se c'è "V" se si abbiamo φ V ψ
if formula.root.value == "V":
    new_foglie = []
    for node in foglie:
        ins_formula_sx = []
        ins_formula_dx = []
        # crea un nodo della formula φ
        phi = formula.root.children[0]
        ins_formula_sx.append(Formula(phi))
        # crea un nodo della formula ψ
        psi = formula.root.children[1]
        ins_formula_dx.append(Formula(psi))
        # crea un nodo del tableaux che ha come formula φ
        new_nodo_sx = self.Nodo(ins_formula_sx)
        # crea un nodo del tableaux che ha come formula ψ
        new_nodo_dx = self.Nodo(ins_formula_dx)
        node.children.append(new_nodo_sx)
        node.children.append(new_nodo_dx)
        new_foglie.append(new_nodo_sx)
        new_foglie.append(new_nodo_dx)
    foglie = new_foglie
```

The implementation

$$\frac{\phi \to \psi}{\neg\phi \mid \psi}$$

```
# controllo se c'è "→" se si abbiamo φ → ψ
if formula.root.value == "→":
    new_foglie = []
    for node in foglie:
        ins_formula_sx = []
        ins_formula_dx = []
        # crea un nodo della formula ~φ
        phi = Node("~")
        phi.children.append(formula.root.children[0])
        ins_formula_sx.append(Formula(phi))
        # crea un nodo della formula ψ
        psi = formula.root.children[1]
        ins_formula_dx.append(Formula(psi))
        # crea un nodo del tableaux che ha come formula ~φ
        new_nodo_sx = self.Nodo(ins_formula_sx)
        # crea un nodo del tableaux che ha come formula ψ
        new_nodo_dx = self.Nodo(ins_formula_dx)
        node.children.append(new_nodo_sx)
        node.children.append(new_nodo_dx)
        new_foglie.append(new_nodo_sx)
        new_foglie.append(new_nodo_dx)
    foglie = new_foglie
```

The implementation

The last check is for truth value assignments. If I have a negation of a term I check that it does not already have a value of True, in which case the branch would be closed, otherwise False is assigned.

```
# 4° CONTROLLO: TERMINI
for formula in nodo.ins_formula:
    if formula.root.value == "~":
        if formula.root.children[0].estTermine():
            termine = formula.root.children[0]
            if termine.boolean == True:
                # se il valore assegnato al termine è true avrei una contraddizione e perciò devo chiudere la branch
                nodo.children = [self.Nodo([Formula(Node("X"))])]
                return
            else:
                # non ho assegnato nessun valore di verità al termine e quindi posso dargli False
                termine.boolean = False
                # controllo una chiusura positiva della branch, che può avvenire se mi trovo in una foglia del tableaux
                if len(nodo.children) == 0:
                    count = 0
                    # controlla i boolean dei termini di ins_formula
                    for term in self.terms:
                        # se sono tutti diversi da None return true
                        if term.boolean is not None:
                            count += 1
                    if count == self.terms_len:
                        self.ret = True
                        nodo.children = [self.Nodo([Formula(Node("O"))])]
                        return
```

The implementation

If I have only one term I verify that it does not already have value False, in which case the branch would be closed, otherwise True is assigned.

```
if formula.root.estTermine():
    termine = formula.root
    # se il valore assegnato al termine è false avrei una contraddizione e perciò devo chiudere la branch
    if termine.boolean == False:
        nodo.children = [self.Nodo([Formula(Node("X"))])]
        return
    else:
        # non ho assegnato nessun valore di verità al termine e quindi posso dargli True
        termine.boolean = True
        if len(nodo.children) == 0:
            count = 0
            # controlla i boolean dei termini di ins_formula
            for term in self.terms:
                # se sono tutti diversi da None return true
                if term.boolean is not None:
                    count += 1
            if count == self.terms_len:
                self.ret = True
                nodo.children = [self.Nodo([Formula(Node("O"))])]
                return
```

The implementation

Example of the tableau tree built through our implementation:

$$\neg((p \to q) \land (p \land q \to r) \to (p \to r))$$

```
Rappresentazione dell'albero tableaux:

['~(((p → q) ∧ ((p ∧ q) → r)) → (p → r))']
    ['((p → q) ∧ ((p ∧ q) → r))', '~(p → r)']
        ['(p → q)', '((p ∧ q) → r)']
            ['p', '~r']
                ['~p']
                    ['X']
                ['q']
                    ['~(p ∧ q)']
                        ['~p']
                            ['X']
                        ['~q']
                            ['X']
                    ['r']
                        ['X']

La formula non è soddisfacibile
```