# TABLEAUX - SATISFIABILITY

LOGIC FOR ARTIFICIAL INTELLIGENCE SEMINAR

GAUDIANO ANTONIO

## THE UNDERLYING THEORY

## THEORY: TABLEAU CALCULATIONS

Logical computations are "formal systems," that is, systems that operate at a syntactic level, which make it possible to establish some semantic property (validity/satisfiability/unsatisfiability) of the formulas of a logic.

Tableau calculations make it possible to determine whether or not a formula is unsatisfiable (if its negation is valid).

## THEORY: RULES

Tableau calculations consist of rules that operate on sets of formulas and that allow formulas to be “deconstructed" in the set. Tableau calculus for classical propositional logic consists of rules for deconstructing formulas that are:

- nonnegated composites: $A \land B$, $A \lor B$, $A \to B$;
- negated composites: $\neg(A \land B)$, $\neg(A \lor B)$, $\neg(A \to B)$;
- doubly negated: $\neg\neg A$.

## THEORY: RULES - EXAMPLE

$$\frac{\Gamma, A \wedge B}{\Gamma, A, B}\,\wedge \qquad\qquad \frac{\Gamma, \neg(A \wedge B)}{\Gamma, \neg A \mid \Gamma, \neg B}\,\neg\wedge$$

- Rules act on sets of formulas, read from top to bottom, and have a name indicated to the right of the horizontal line (which we will often avoid to indicate).
- The set above the line is called the premise; in the premise the formula on which the rule acts is highlighted, which is called the main formula of the rule.

## THEORY: EXPANSION RULES

In order to formalize tableau in a compact way we introduce a classification of rules that operates on binary connectives, based on the number of conclusions.

$$\frac{\Gamma, A \wedge B}{\Gamma, A, B}\,{\scriptstyle\wedge} \qquad \frac{\Gamma, \neg(A \vee B)}{\Gamma, \neg A, \neg B}\,{\scriptstyle\neg\vee} \qquad \frac{\Gamma, \neg(A \to B)}{\Gamma, A, \neg B}\,{\scriptstyle\neg\to}$$

Alpha-rules, 1 conclusion

$$\frac{\Gamma, \neg(A \wedge B)}{\Gamma, \neg A \mid \Gamma, \neg B}\,{\scriptstyle\neg\wedge} \qquad \frac{\Gamma, A \vee B}{\Gamma, A \mid \Gamma, B}\,{\scriptstyle\vee} \qquad \frac{\Gamma, A \to B}{\Gamma, \neg A \mid \Gamma, B}\,{\scriptstyle\to}$$

Beta-rules, 2 conclusions

# THEORY: RULES INTERPRETATION

- **Rule with one conclusion:** The tree node corresponding to the premise of the rule has as its only child the node corresponding to the conclusion.
- **Rule with two conclusions:** The node of the tree corresponding to the premise of the rule has two children corresponding to the two conclusions of the rule.

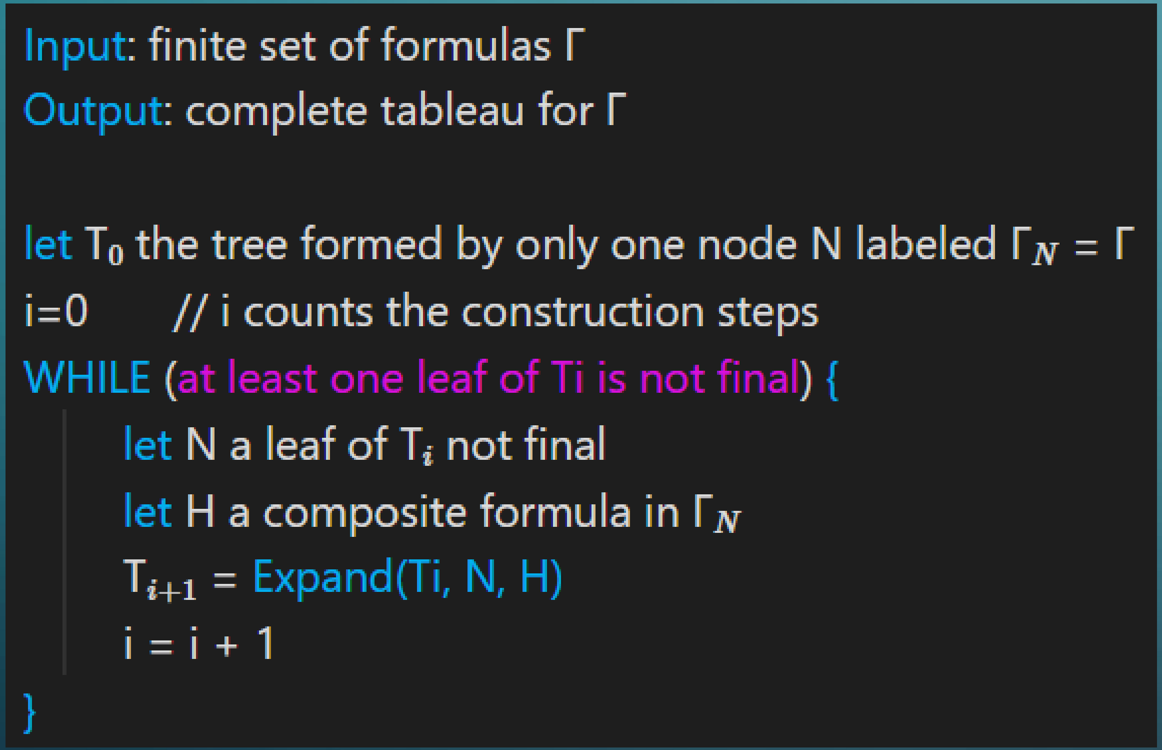
$$\frac{\Gamma, A \wedge B}{\Gamma, A, B}\wedge \quad \Longrightarrow \quad \begin{tikzcd}[arrows=-] \Gamma, A \wedge B \arrow[d] \\ \Gamma, A, B \end{tikzcd} \qquad \frac{\Gamma, A \vee B}{\Gamma, A \mid \Gamma, B}\vee \quad \Longrightarrow \quad \begin{tikzcd}[arrows=-] & \Gamma, A \vee B \arrow[dl] \arrow[dr] & \\ \Gamma, A & & \Gamma, B \end{tikzcd}$$

## THEORY: BUILDING A TABLEAU (INTUITIVE)

A proof tree (a tableau) is obtained by repeatedly applying the rules of the calculus from a set of formulas (we will see a formal definition later).

$$\dfrac{\dfrac{\neg((p \to q) \to (\neg q \to \neg p))}{(p \to q), \neg(\neg q \to \neg p)}\,{\scriptstyle \neg\to}}{\dfrac{\dfrac{\neg p, \neg(\neg q \to \neg p)}{\neg p, \neg q, \neg\neg p}\,{\scriptstyle \neg\to}}{\neg p, \neg q, p}\,{\scriptstyle \neg\neg} \quad\Bigg|\quad \dfrac{\dfrac{q, \neg(\neg q \to \neg p)}{q, \neg q, \neg\neg p}\,{\scriptstyle \neg\to}}{q, \neg q, p}\,{\scriptstyle \neg\neg}}\,{\scriptstyle \to}$$

## THEORY: BUILDING A TABLEAU - ALGORITHM



```
Input: finite set of formulas $\Gamma$
Output: complete tableau for $\Gamma$

let $T_0$ the tree formed by only one node N labeled $\Gamma_N = \Gamma$
i=0      // i counts the construction steps
WHILE (at least one leaf of Ti is not final) {
    let N a leaf of $T_i$ not final
    let H a composite formula in $\Gamma_N$
    $T_{i+1}$ = Expand(Ti, N, H)
    i = i + 1
}
```

- Expand(Ti, N, H) performs an expansion step of the Ti tree, breaking down the formula H of the node N, returning the Ti+1 tree obtained in this way.
- The expansion is defined according to the form of the rules we have seen before.

## THEORY: BUILDING A TABLEU - EXPANSION

- If H = ¬¬A, then the Ti+1 tree is built by adding a child N' as successor of N with associated set: $\Gamma N' = (\Gamma N \setminus \{\neg\neg A\}) \cup \{A\}$.
- If H is an alpha-formula, then the Ti+1 tree is built by adding a child N' as successor of N with associated set: $\Gamma N' = (\Gamma N \setminus \{H\}) \cup \{H1, H2\}$.
- If H is a beta-formula, then the Ti+1 tree is built by adding two child N' and N'' as successors of N with associated set:
  - $\Gamma N' = (\Gamma N \setminus \{H\}) \cup \{H1\}$
  - $\Gamma N'' = (\Gamma N \setminus \{H\}) \cup \{H2\}$

Note: H1 and H2 are the 'reduced' formulas.

## THEORY: BUILDING A TABLEAU - NOTES

- The algorithms always terminates, if not an infinite long branch will be constructed and this is not possible since we start with a finite set of formula. (note that a rigorous proof exists but is out of the scope for this presentation)

- The algorithm is not deterministic, at each iteration we have to choose:
  - The leaf N we want to expand among all the non final ones
  - The composite formula we want to operate with among all the ones in $\Gamma N$

## THEORY: SATISFIABILITY

If a set of formulas Γ has a complete open tableau we can extract an evaluation from it that satisfies Γ, so Γ is satisfiable.

A tableau is complete if the set associated with each leaves contains only literals.

A tableau is said to be open if it has at least one open branch.

## THEORY: COMPLETENESS THEOREM

<u>A finite set of formulas Γ is unsatisfiable if and only if exists a complete closed tableau for Γ.</u>

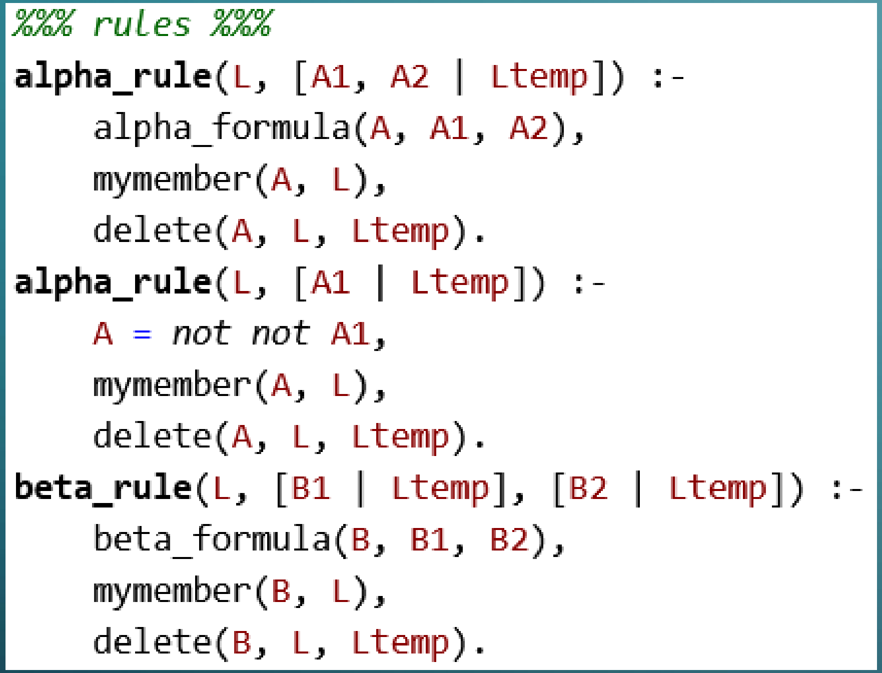
A tableau is closed when all the branches are closed.

A branch is said to be closed if contains a complementary pair of literals (A, ¬A).

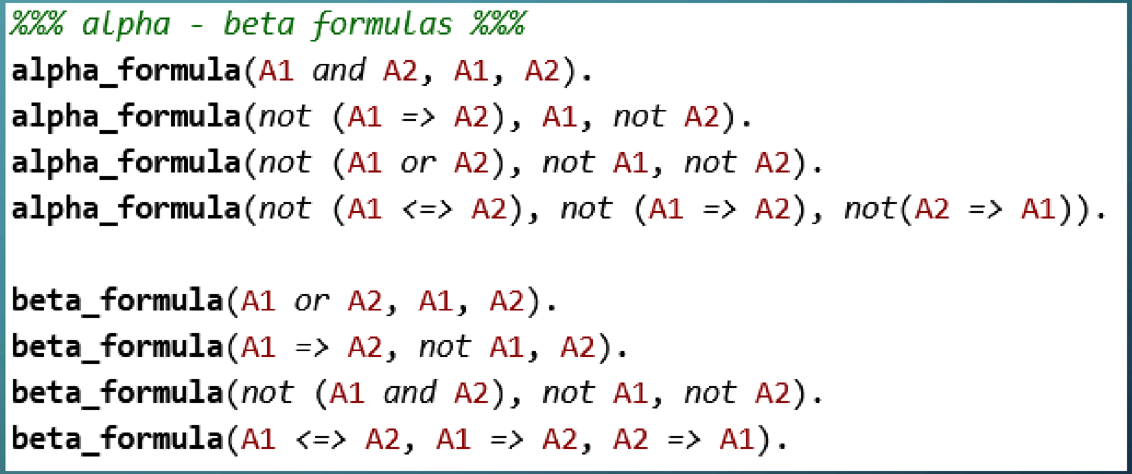
<u>Closed tableau for {¬H} ⇔ ¬H is unsatisfiable ⇔ H is valid</u>

## CODE

## CODE: EXPANSION RULES

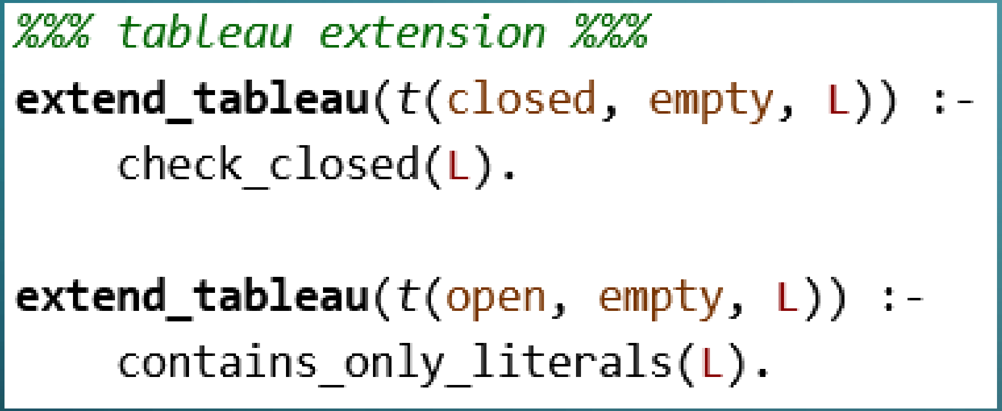


```
%%% rules %%%
alpha_rule(L, [A1, A2 | Ltemp]) :-
    alpha_formula(A, A1, A2),
    mymember(A, L),
    delete(A, L, Ltemp).
alpha_rule(L, [A1 | Ltemp]) :-
    A = not not A1,
    mymember(A, L),
    delete(A, L, Ltemp).
beta_rule(L, [B1 | Ltemp], [B2 | Ltemp]) :-
    beta_formula(B, B1, B2),
    mymember(B, L),
    delete(B, L, Ltemp).
```

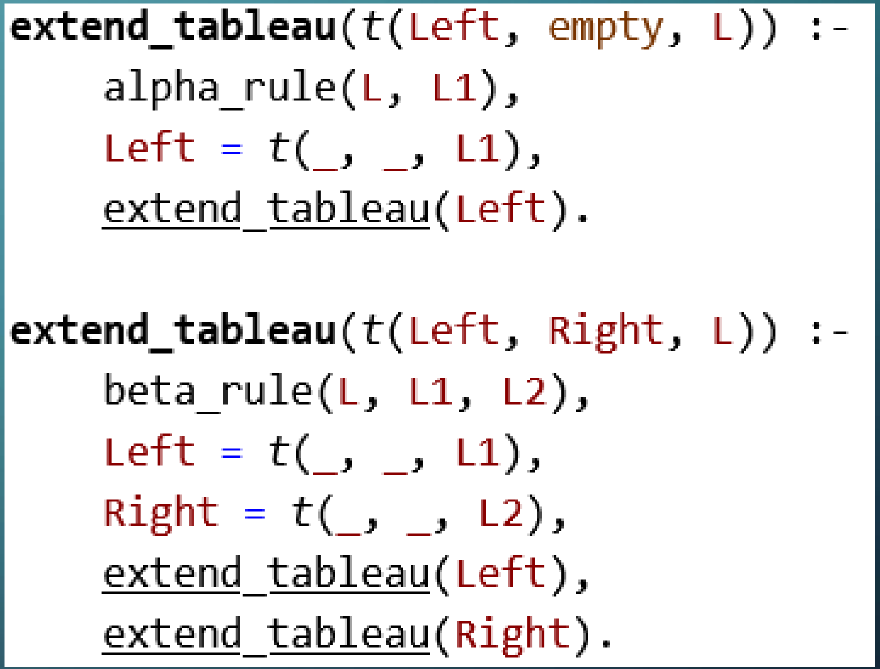


```
%%% alpha - beta formulas %%%
alpha_formula(A1 and A2, A1, A2).
alpha_formula(not (A1 => A2), A1, not A2).
alpha_formula(not (A1 or A2), not A1, not A2).
alpha_formula(not (A1 <=> A2), not (A1 => A2), not(A2 => A1)).

beta_formula(A1 or A2, A1, A2).
beta_formula(A1 => A2, not A1, A2).
beta_formula(not (A1 and A2), not A1, not A2).
beta_formula(A1 <=> A2, A1 => A2, A2 => A1).
```

## CODE: TABLEAU EXPANSION

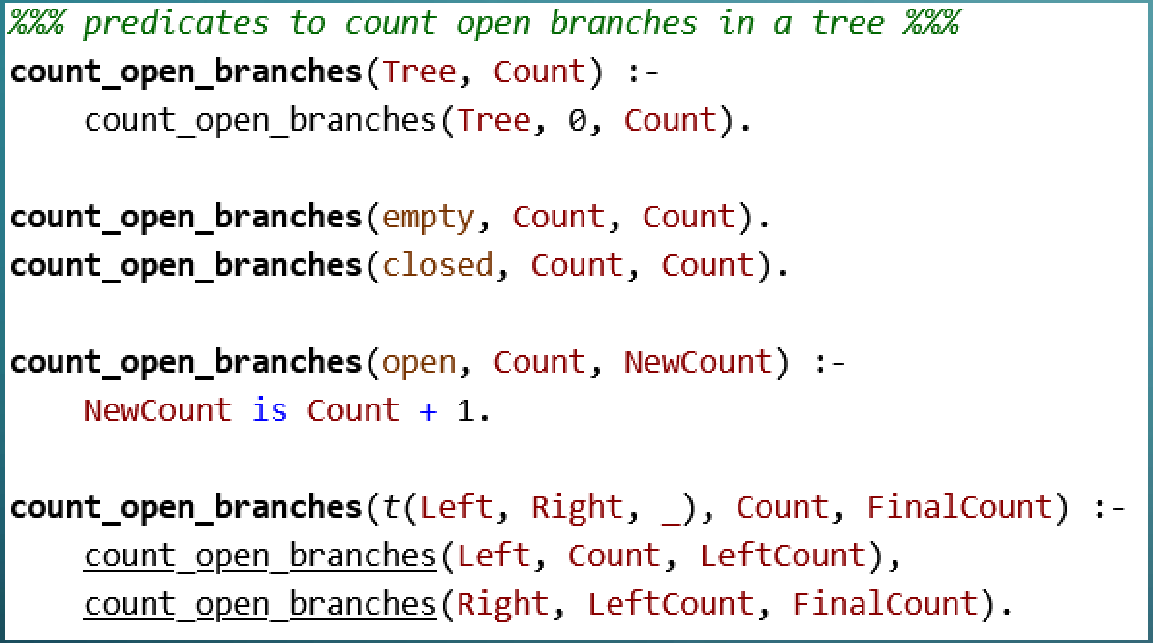


```
%%% tableau extension %%%
extend_tableau(t(closed, empty, L)) :-
    check_closed(L).

extend_tableau(t(open, empty, L)) :-
    contains_only_literals(L).
```



```
extend_tableau(t(Left, empty, L)) :-
    alpha_rule(L, L1),
    Left = t(_, _, L1),
    extend_tableau(Left).

extend_tableau(t(Left, Right, L)) :-
    beta_rule(L, L1, L2),
    Left = t(_, _, L1),
    Right = t(_, _, L2),
    extend_tableau(Left),
    extend_tableau(Right).
```

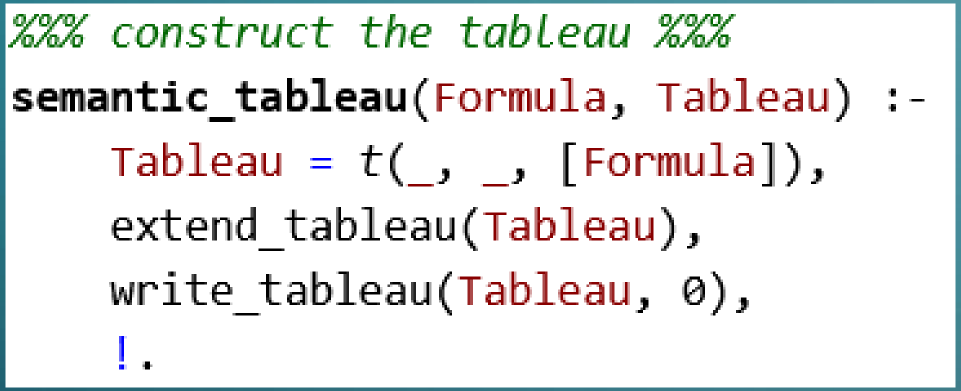
# CODE: IS THE TABLEAU CLOSED?



```
%%% predicates to count open branches in a tree %%%
count_open_branches(Tree, Count) :-
    count_open_branches(Tree, 0, Count).

count_open_branches(empty, Count, Count).
count_open_branches(closed, Count, Count).

count_open_branches(open, Count, NewCount) :-
    NewCount is Count + 1.

count_open_branches(t(Left, Right, _), Count, FinalCount) :-
    count_open_branches(Left, Count, LeftCount),
    count_open_branches(Right, LeftCount, FinalCount).
```

We simply check if there are open branches (and how many).

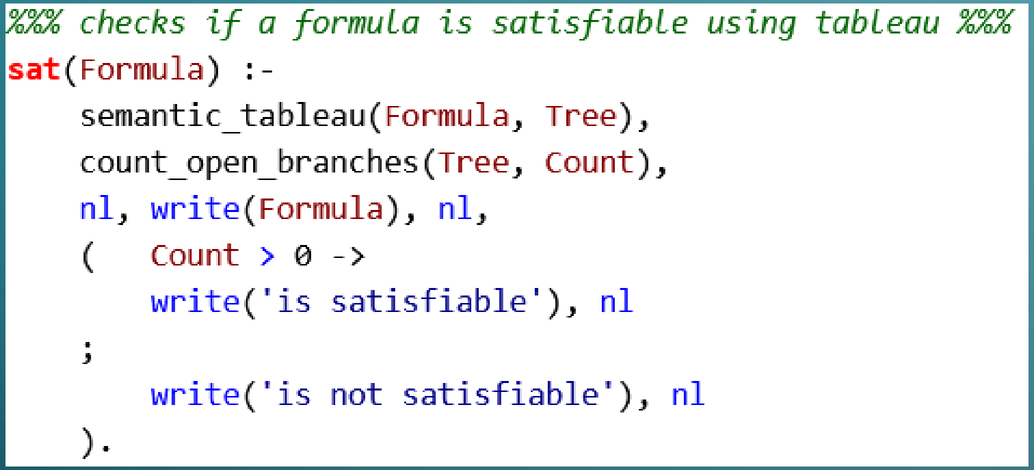
## CODE: BUILDING THE TABLEAU



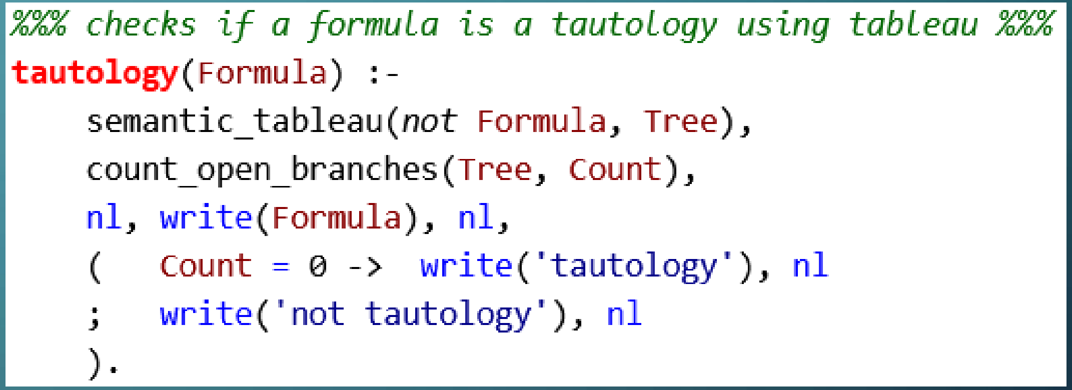
```
%%% construct the tableau %%%
semantic_tableau(Formula, Tableau) :-
    Tableau = t(_, _, [Formula]),
    extend_tableau(Tableau),
    write_tableau(Tableau, 0),
    !.
```

Note the presence of the cut operator: since the algorithm is not deterministic many possible tableau can be found, we are interested only in one of them.

## CODE: SATISFIABILITY



```prolog
%%% checks if a formula is satisfiable using tableau %%%
sat(Formula) :-
    semantic_tableau(Formula, Tree),
    count_open_branches(Tree, Count),
    nl, write(Formula), nl,
    (   Count > 0 ->
        write('is satisfiable'), nl
    ;
        write('is not satisfiable'), nl
    ).
```



```prolog
%%% checks if a formula is a tautology using tableau %%%
tautology(Formula) :-
    semantic_tableau(not Formula, Tree),
    count_open_branches(Tree, Count),
    nl, write(Formula), nl,
    (   Count = 0 ->  write('tautology'), nl
    ;   write('not tautology'), nl
    ).
```

## CODE: OTHER PREDICATES

In the code there are other ‘utility’ predicates such as:

- Check_closed(L)
- Contains_only_literals([])
- Write_tableau(Tableau, 0)
- Mymember(X, [])
- Delete(X, [])
- Write_multiple_chars(Char, N)

THANK YOU FOR THE ATTENTION