Tree Method for Establishing Satisfiability

Robert Keller

14 February 2005

Satisfaction

- An assignment of truth values to proposition symbols in a formula that induces the value T (true) is said to **satisfy** the formula.
- If there is **at least one** such assignment, the formula is called **satisfiable**.
- This is weaker than being a tautology, in which case **all** assignments must satisfy the formula.

Examples

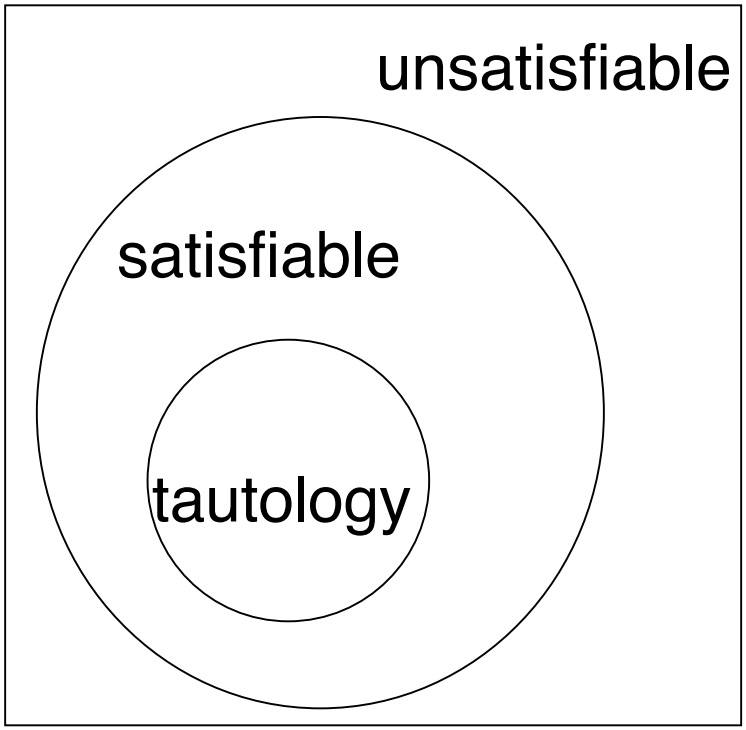
$p \vee \neg p$ is a tautology

$p \vee q$ is satisfiable, but not a tautology

$p \wedge \neg p$ is not satisfiable

$\bot$ is not satisfiable

Trichotomy



More Examples to Classify

- $(p \rightarrow q) \rightarrow (q \rightarrow p)$
- $(p \rightarrow q) \wedge \neg(q \rightarrow p)$
- $(p \rightarrow q) \wedge \neg(p \rightarrow q)$
- $(p \rightarrow q) \wedge (q \rightarrow p)$
- $(p \rightarrow q) \vee (q \rightarrow p)$
- $(p \rightarrow q) \vee \neg(p \rightarrow q)$

Fact

- Formula φ is a tautology
  iff
  $\neg\varphi$ is *not* satisfiable.

- Obvious since an assignment that induces T in φ must induce F in $\neg\varphi$ and vice-versa.

- (If **all** assignments induce T in φ, then all assignments induce F in $\neg\varphi$.)

More Useful Facts

- $\varphi \wedge \psi$ is a tautology iff φ and ψ both are.
- $\varphi \vee \psi$ is satisfiable iff one of them is.
- $\varphi \wedge \psi$ is not necessarily satisfiable even if both φ and ψ are.
- $\varphi \vee \psi$ may be a tautology even if neither φ nor ψ is.

Consistency

- Another term used for satisfiability is that the formula is **consistent**.

- Moreover, an entire **set** of formulas is called **consistent** if there is at least one assignment that satisfies them all **simultaneously**.

Example

- $\{\neg p \vee q, \neg q \vee r, p, r\}$ is consistent
- $\{\neg p \vee q, \neg q \vee r, p, \neg r\}$ is inconsistent

Fact

- An implication of the form

$$(\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n) \rightarrow \psi$$

is a tautology iff the set of formulas

$$\{\varphi_1, \varphi_2, \ldots, \varphi_n, \neg\psi\}$$

is inconsistent.

Algorithmic Implications

- If we have an algorithm to determine satisfiability, we can easily use it to get one to determine tautology (just use it on $\neg\varphi$).

- If we have a complete set of derivation rules, then a **set** of formulas is inconsistent iff we can derive $\perp$ from it.

The Tree Method for Satisfiability

- The tree method is an easy to use method for determining whether or not a propositional formula is satisfiable.
- The method also extends to **predicate logic** as a partial decision procedure.
- The method has a superficial resemblance to Quine's (or Boole/Shannon) method, but is not the same.

Historical

- The method also goes by the name “semantic tableau” or “analytic tableau”.
- The first thorough treatment is in Raymond M. Smullyan, *First Order Logic*, Springer Verlag, 1971. However, this treatment can be rather cryptic in places.
- Smullyan attributes the origin of the method to Beth and Hintikka, and ultimately Gentzen.
- A more readable treatment is in Joseph Bessie and Stuart Glennan, *Elements of Deductive Inference*, Wadsworth Publishing, 2000.

The method

- Start with a formula the satisfiability of which is to be checked.
- This formula is the **root** of the tree.
- Grow the tree downward in a manner to be shown.
- The original formula is unsatisfiable iff all paths from root to a leaf ultimately become "**closed**".

Closure

- A path is **closed** if some formula appears on the path both negated and un-negated.
- A tree is **closed** if all of its paths from root to leaves are closed.
- If at completion, any **non-closed** path corresponds to one or more assignments that satisfy the original formula.

Growing the tree

- As formulas are used, the become “retired” by placing a check by them.
- Formulas are retired from the upper parts of the tree.
- Each retirement of a formula introduces formulas that are **shorter** into all open paths below.
- The fact that introduced formulas are shorter ensures that the process eventually terminates.

Tree Rules: $\varphi \wedge \psi$

- This formula is retired and φ and are added to the tree ψ.

$\varphi \wedge \psi$ ✓ check indicates retirement

φ

ψ

} replace with

Meaning: Both sub-formulas have to be satisfied simultaneously for the original to be satisfied.

Example: $p \wedge \neg q$

- This formula is retired and φ and are added to the tree ψ.

$$p \wedge \neg q \checkmark$$

$$p$$

$$\neg q$$

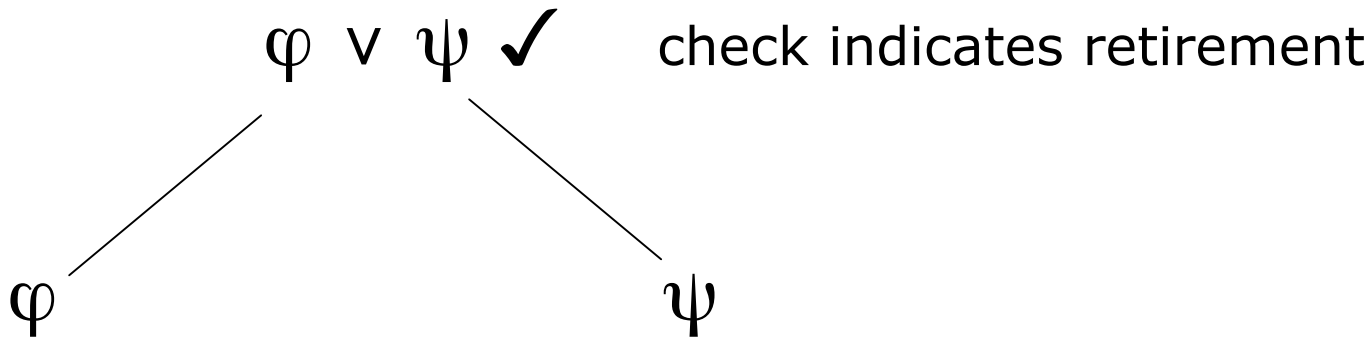
This construction is **complete**.
There is only one path and it is **not closed**.
An assignment satisfying the original is p = T, q = F.

Termination

- The construction is complete when every unchecked formula is a literal (single proposition symbol or its negation).

Tree Rules: $\varphi \vee \psi$

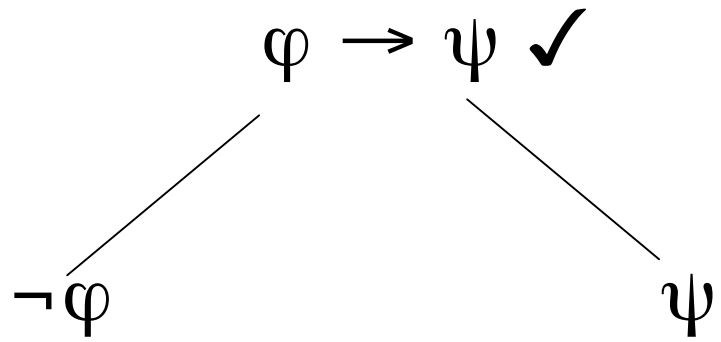
- This formula is retired and the tree **branches** to φ and ψ:



Meaning: At least one sub-formula has to be satisfied for the original to be satisfied.

Tree Rules: $\varphi \rightarrow \psi$

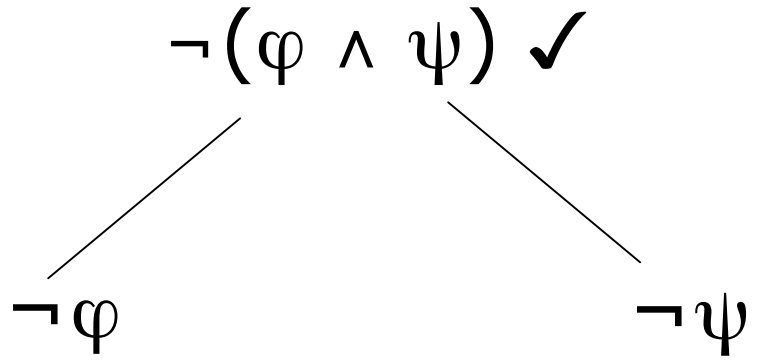
- This formula is retired and the tree **branches**, with φ negated:



(Recall that $\varphi \rightarrow \psi$ is equivalent to $\neg\varphi \vee \psi$.).

Tree Rules: $\neg(\varphi \wedge \psi)$

- This formula is retired and the tree branches.



(Recall that $\neg(\varphi \wedge \psi)$ is equivalent to $\neg\varphi \vee \neg\psi$.).

Tree Rules: $\neg(\varphi \vee \psi)$

- This formula is retired and $\neg\varphi$ and $\neg\psi$ are added to the tree without branching.

$$\neg(\varphi \vee \psi) \checkmark$$
$$\neg\varphi$$
$$\neg\psi$$

(Recall that $\neg(\varphi \vee \psi)$ is equivalent to $\neg\varphi \wedge \neg\psi$.).

Tree Rules: $\neg(\varphi \rightarrow \psi)$

- This formula is retired and φ and $\neg\psi$ are added to the tree without branching.

$$\neg(\varphi \rightarrow \psi) \checkmark$$

$$\varphi$$

$$\neg\psi$$

Tree Rules: $\neg\neg\varphi$

- This formula is retired and φ and is added to the tree without branching.

$\neg\neg\varphi$ ✓

φ

Tree Rules: Literals

- Literals (φ or $\neg\varphi$, where φ is a proposition symbol) are left as is.

Example tree construction

$$((p \wedge q) \vee (\neg q \wedge \neg p))$$

Example tree construction

$((p \wedge q) \vee (\neg q \wedge \neg p))$ ✓

$p \wedge q$

$\neg q \wedge \neg p$

Example tree construction

$((p \wedge q) \vee (\neg q \wedge \neg p))$ ✓

$p \wedge q$ ✓

p

q

$\neg q \wedge \neg p$ ✓

$\neg q$

$\neg p$

Neither path is closed.

Example tree construction

$$\neg((p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p))$$

Example tree construction

$\neg((p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p))$ ✓

$(p \rightarrow q)$

$\neg(\neg q \rightarrow \neg p)$

Example tree construction

$\neg((p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p))$ ✓

$(p \rightarrow q)$

$\neg(\neg q \rightarrow \neg p)$ ✓

$\neg q$

$\neg \neg p$

Example tree construction

$\neg((p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p))$ ✓

$(p \rightarrow q)$

$\neg(\neg q \rightarrow \neg p)$ ✓

$\neg q$

$\neg\neg p$ ✓

p

Example tree construction

$\neg((p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p))$ ✓

$(p \rightarrow q)$ ✓

$\neg(\neg q \rightarrow \neg p)$ ✓

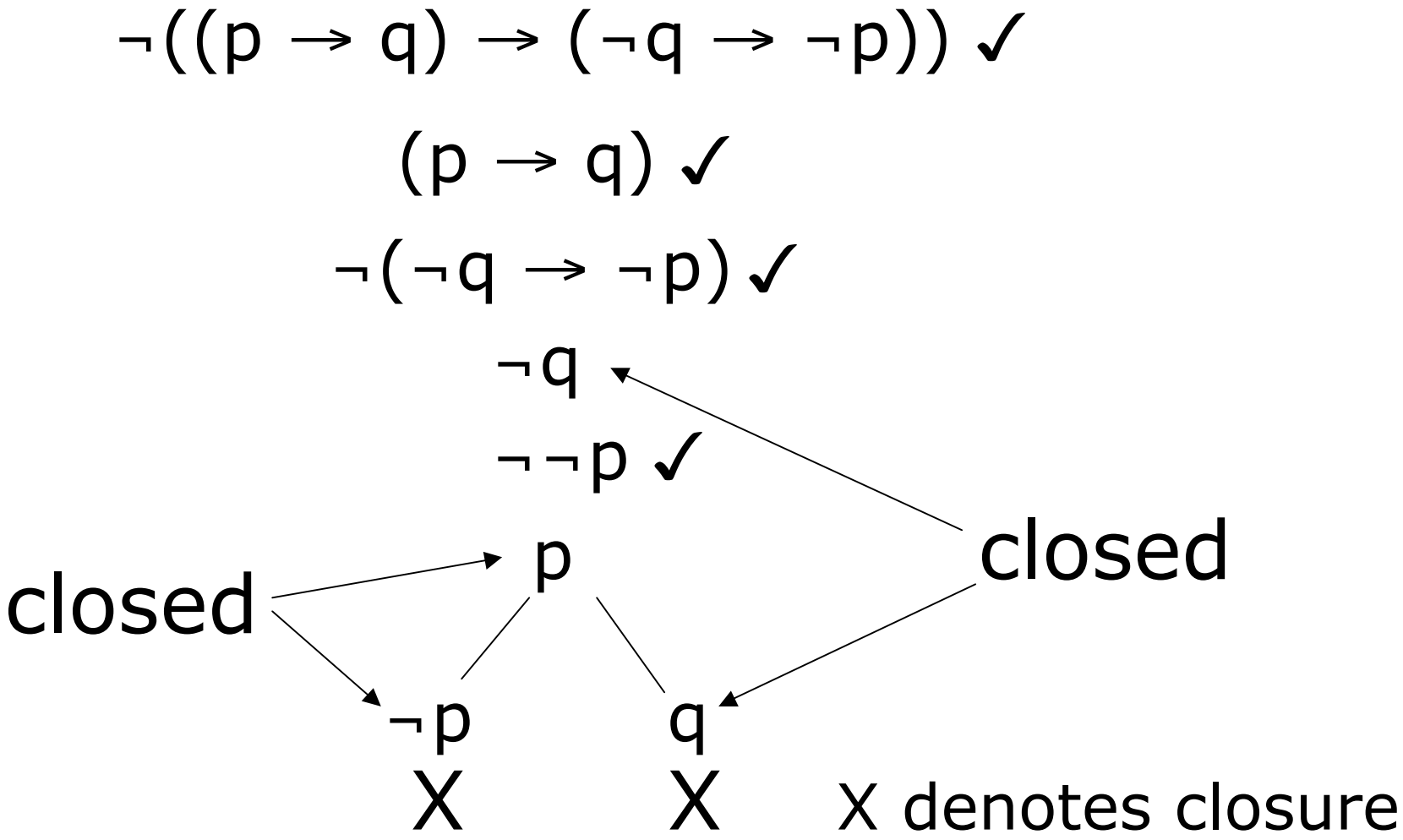
$\neg q$

$\neg\neg p$ ✓

p

$\neg p$ $\quad$ q

Example tree construction



Meaning

- The original formula is **unsatisfiable** iff all paths is closed when the tree construction stops.

- So if the original formula is the **negation** of a formula to be tested for being a tautology, the **unnegated** formula is a tautology iff all paths are closed.

Example tree construction

$$\neg(q \rightarrow (\neg q \rightarrow \neg p))$$

Example tree construction

$\neg(q \rightarrow (\neg q \rightarrow \neg p))$ ✓

q

$\neg(\neg q \rightarrow \neg p)$

Example tree construction

$\neg(q \rightarrow (\neg q \rightarrow \neg p))$ ✓

q

$\neg(\neg q \rightarrow \neg p)$ ✓

$\neg q$

$\neg\neg p$

X

closed

Example

$$\neg((\neg p \wedge \neg q) \rightarrow (\neg q \vee \neg p))$$

Example

$\neg((\neg p \wedge \neg q) \rightarrow \neg(q \vee p))$ ✓

$(\neg p \wedge \neg q)$

$\neg\neg(q \vee p)$

Example

$\neg((\neg p \wedge \neg q) \rightarrow \neg(q \vee p))$ ✓

$(\neg p \wedge \neg q)$

$\neg\neg(q \vee p)$ ✓

$(q \vee p)$

Example

$\neg((\neg p \wedge \neg q) \rightarrow \neg(q \vee p))$ ✓

$(\neg p \wedge \neg q)$ ✓

$\neg\neg(q \vee p)$ ✓

$(q \vee p)$

$\neg p$

$\neg q$

Example

$\neg((\neg p \wedge \neg q) \rightarrow \neg(q \vee p))$ ✓

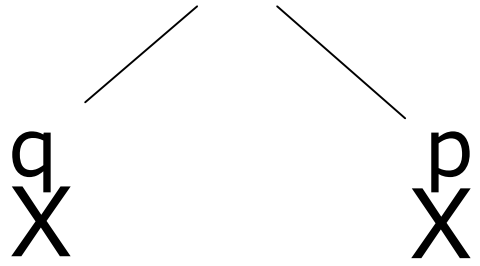
$(\neg p \wedge \neg q)$ ✓

$\neg\neg(q \vee p)$ ✓

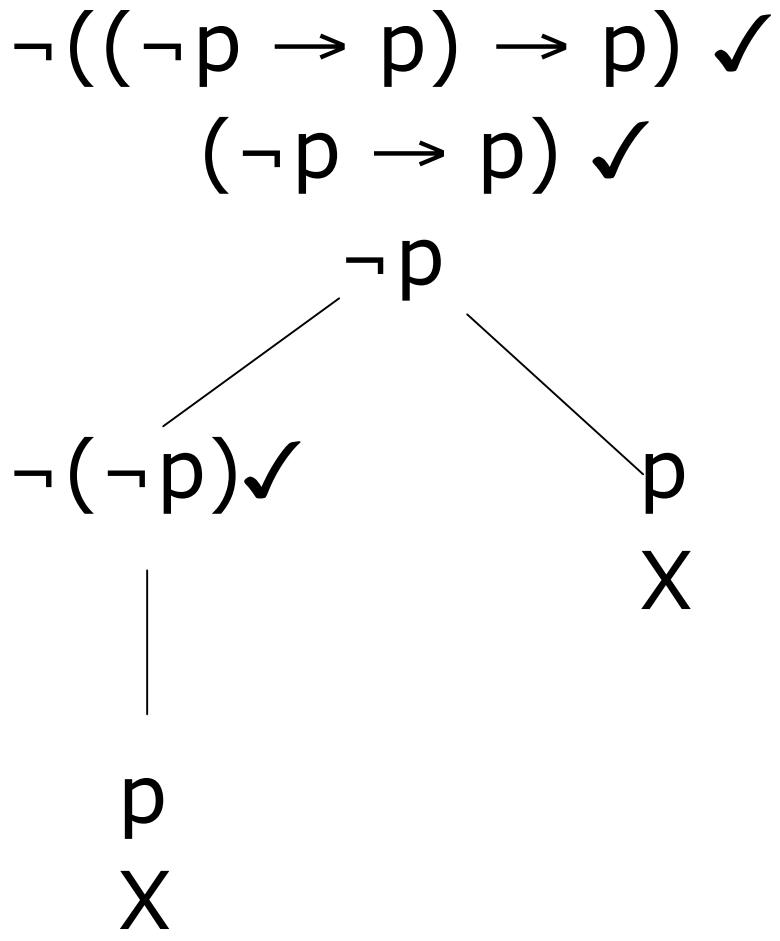
$(q \vee p)$ ✓

$\neg p$

$\neg q$



Example



Early Closure

- We don’t have to wait until a literal and its complement appear on a path to get closure.
- If **any** formula and its complement appear on the same path, this is sufficient for closure.

Example

$$\neg((\neg p \wedge \neg q) \rightarrow (\neg p \wedge \neg q))$$

Example

$\neg((\neg p \wedge \neg q) \rightarrow (\neg p \wedge \neg q))$ ✓

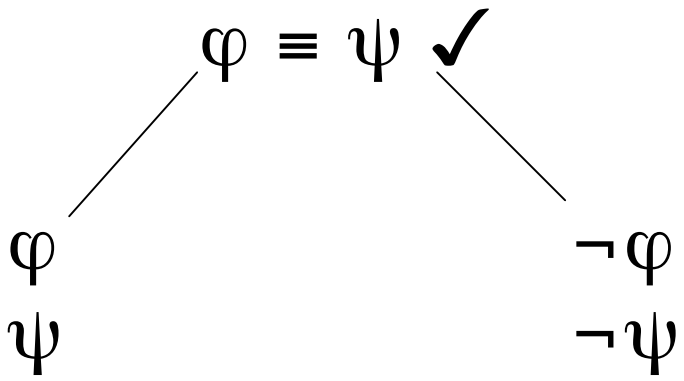
$(\neg p \wedge \neg q)$

$\neg(\neg p \wedge \neg q)$

X

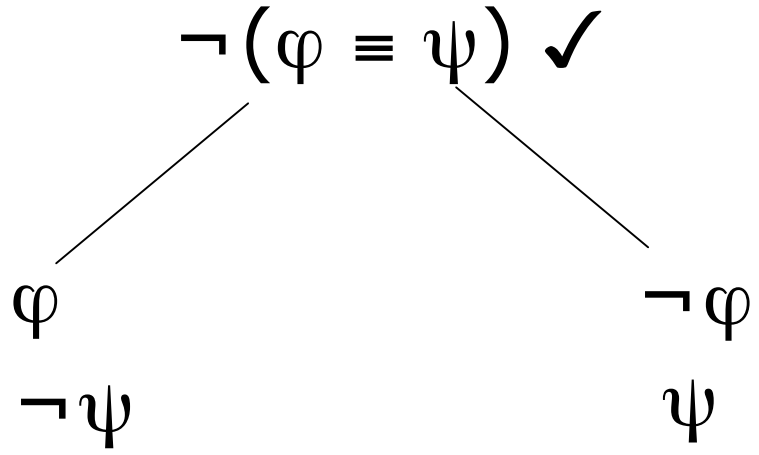
Additional Tree Rules: $\varphi \equiv \psi$

- This formula is retired and the tree **branches** with the formulas and their negations stacked:



Additional Tree Rules: $\neg(\varphi \equiv \psi)$

- This formula is retired and the tree **branches**, with φ negated:



Propositional Tree Rule Summary

stack	$\varphi \wedge \psi$ φ ψ	$\neg(\varphi \vee \psi)$ $\neg\varphi$ $\neg\psi$	$\neg(\varphi \rightarrow \psi)$ φ $\neg\psi$		$\neg\neg\varphi$ φ
branch	$\varphi \vee \psi$ φ $\quad$ ψ	$\neg(\varphi \wedge \psi)$ $\neg\varphi$ $\quad$ $\neg\psi$	$\varphi \rightarrow \psi$ $\neg\varphi$ $\quad$ ψ	$(\varphi \equiv \psi)$ φ $\neg\varphi\psi$ $\neg\psi$ $\neg(\varphi \equiv \psi)$ φ $\neg\varphi\neg\psi$ ψ	

Soundness and Completeness

- We assert, without proof, that the tree method is sound and complete for proposition logic.

The Tree Method is Easy to Implement (1-page implementation in Prolog)

```
isTautology(Formula) :-
            refute(not(Formula)).

refute(Formula) :- refute([Formula], []).

refute([Formula | _Formulas], Path) :-
    member(not(Formula), Path), !.

refute([not(Formula) | _Formulas], Path) :-
    member(Formula, Path), !.

refute([Formula | Formulas], Path) :-
    literal(Formula),
    refute(Formulas, [Formula | Path]).

refute([not(not(Formula)) | Formulas], Path) :-
    refute([Formula | Formulas], Path).

refute([and(F, G) | Formulas], Path) :-
    refute([F, G | Formulas], Path).
```

```
refute([or(F, G) | Formulas], Path) :-
    refute([F | Formulas], Path),
    refute([G | Formulas], Path).

refute([implies(F, G) | Formulas], Path) :-
    refute([or(not(F), G) | Formulas], Path).

refute([iff(F, G) | Formulas], Path) :-
    refute([or(and(F, G), and(not(F), not(G))) | Formulas], Path).

refute([not(and(F, G)) | Formulas], Path) :-
    refute([or(not(F), not(G)) | Formulas], Path).

refute([not(or(F, G)) | Formulas], Path) :-
    refute([and(not(F), not(G)) | Formulas], Path).

refute([not(implies(F, G)) | Formulas], Path) :-
    refute([and(F, not(G)) | Formulas], Path).

refute([not(iff(F, G)) | Formulas], Path) :-
    refute([or(and(F, not(G)), and(not(F), G)) | Formulas], Path).

literal(Formula) :- atomic(Formula).
literal(not(Formula)) :- atomic(Formula).
```

Unsatisfiability for Predicate Logic

- The idea is similar to proposition logic:

Proposition Logic	**Predicate Logic**
tautology	valid formula
assignment	interpretation + assignment
satisfiable	satisfiable

¬∃ rule

$$\neg(\exists \nu)\ \varphi \ \checkmark$$
$$(\forall \nu)\ \neg\varphi$$

¬∀ rule

- ¬∀ rule:

$$\neg(\forall \nu)\ \varphi \ \checkmark$$
$$(\exists \nu)\ \neg\varphi$$

∃ rule

$$(\exists \nu)\ \varphi \ \checkmark$$

$$\varphi[c/\nu]$$

where c is a **new** constant not appearing in the tree.

∀ rule

$$\frac{(\forall \nu)\ \varphi}{\varphi[\tau/\nu]}$$ Does not get a check!!

where τ is any term free to replace ν in φ.

This line can used arbitrarily-many times.

Example

$$\neg((\forall x)p(x) \rightarrow (\exists x)p(x))$$

Example

$\neg((\forall x)p(x) \rightarrow (\exists x)p(x))$ ✓

$(\forall x)p(x)$

$\neg(\exists x)p(x)$

Example

$\neg((\forall x)p(x) \rightarrow (\exists x)p(x))$ ✓

$(\forall x)p(x)$

$\neg(\exists x)p(x)$ ✓

$(\forall x)\ \neg p(x)$

Example

$\neg((\forall x)p(x) \rightarrow (\exists x)p(x))$ ✓

$(\forall x)p(x)$ ∀elim

$\neg(\exists x)p(x)$ ✓

$(\forall x)\ \neg p(x)$ ∀elim

$p(z)$

$\neg p(z)$

X closed

The original formula is not satisfiable.

$(\forall x)p(x) \rightarrow (\exists x)p(x)$ is valid

Example

(in which ∀ rule is used twice on the same line)

$$\neg(\exists x)\,(p(x) \rightarrow (\forall y)p(y))$$

Example

$\neg(\exists x)\ (p(x) \rightarrow (\forall y)p(y))$ ✓

$(\forall x)\ \neg(p(x) \rightarrow (\forall y)p(y))$

Example

$\neg(\exists x)\ (p(x) \rightarrow (\forall y)p(y))$ ✓

$(\forall x)\ \neg(p(x) \rightarrow (\forall y)p(y))$

$\neg(p(z) \rightarrow (\forall y)p(y))$

Example

$\neg(\exists x)\ (p(x) \rightarrow (\forall y)p(y))$ ✓

$(\forall x)\ \neg(p(x) \rightarrow (\forall y)p(y))$

$\neg(p(z) \rightarrow (\forall y)p(y))$ ✓

$p(z)$

$\neg(\forall y)p(y)$

Example

$\neg(\exists x)\ (p(x) \rightarrow (\forall y)p(y))$ ✓

$(\forall x)\ \neg(p(x) \rightarrow (\forall y)p(y))$

$\neg(p(z) \rightarrow (\forall y)p(y))$ ✓

$p(z)$

$\neg(\forall y)p(y)$ ✓

$(\exists y)\ \neg p(y)$

Example

$\neg(\exists x)\,(p(x) \rightarrow (\forall y)p(y))$ ✓

$(\forall x)\,\neg(p(x) \rightarrow (\forall y)p(y))$

$\neg(p(z) \rightarrow (\forall y)p(y))$ ✓

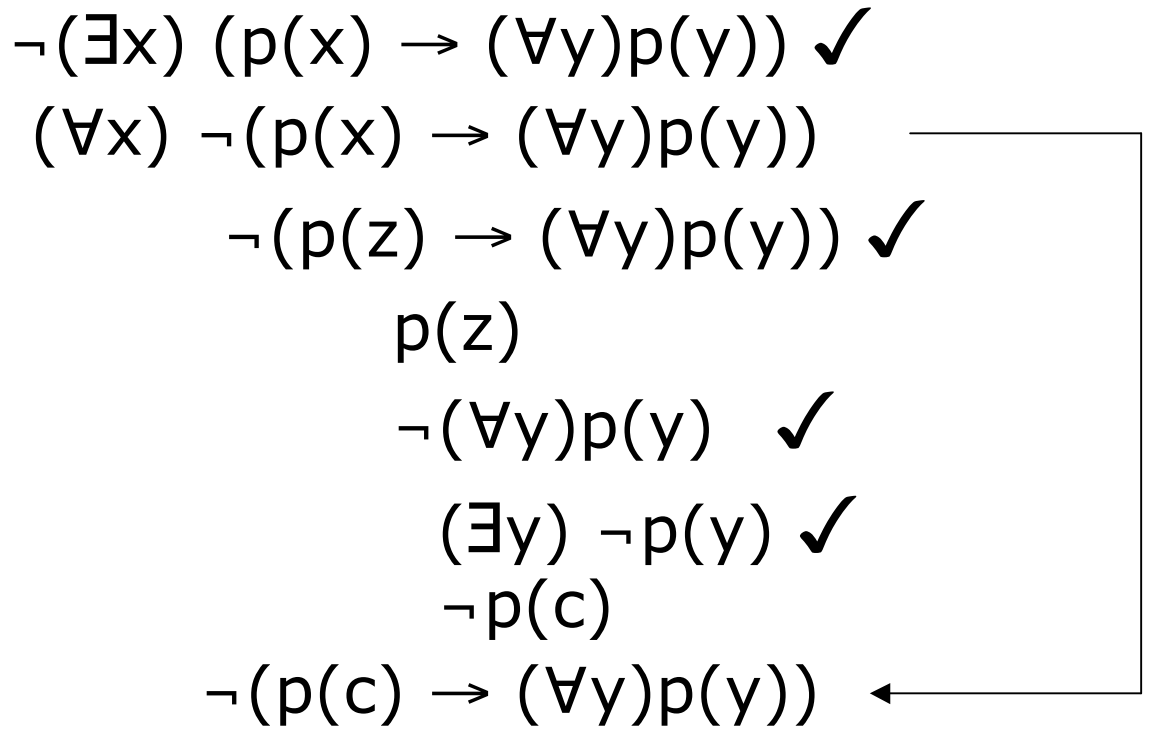
$p(z)$

$\neg(\forall y)p(y)$ ✓

$(\exists y)\,\neg p(y)$ ✓

$\neg p(c)$

Example



Example

$\neg(\exists x)\,(p(x) \rightarrow (\forall y)p(y))$ ✓

$(\forall x)\,\neg(p(x) \rightarrow (\forall y)p(y))$

$\neg(p(z) \rightarrow (\forall y)p(y))$ ✓

$p(z)$

$\neg(\forall y)p(y)$ ✓

$(\exists y)\,\neg p(y)$ ✓

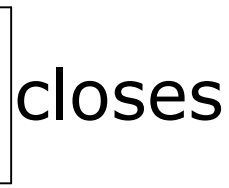
$\neg p(c)$

$\neg(p(c) \rightarrow (\forall y)p(y))$ ✓

$p(c)$

$\neg(\forall y)p(y)$

X



Exercise

- Show that $(\exists x)\,(p(x) \rightarrow (\forall y)p(y))$ is valid (from basic definitions).
- Give a natural deduction proof of it.

Termination

- Unlike the propositional case, the predicate version does not necessarily terminate. This is because the ∀ rule can be used arbitrarily-many times.

- It can be shown, however, that if the original formula is unsatisfiable, then **there exists** a closed tree for it.

Example of Non-Termination

$$(\forall x)\ (\exists y)\ (A(x) \wedge B(y))$$

Example of Non-Termination

$$(\forall x)\ (\exists y)\ (A(x) \wedge B(y))$$

$$(\exists y)\ (A(z) \wedge B(y))$$

Example of Non-Termination

$$(\forall x)\ (\exists y)\ (A(x) \wedge B(y))$$

$$(\exists y)\ (A(z) \wedge B(y)) \checkmark$$

$$(A(z) \wedge B(c_0))$$

Example of Non-Termination

$(\forall x)\ (\exists y)\ (A(x) \wedge B(y))$

$(\exists y)\ (A(z) \wedge B(y))$ ✓

$(A(z) \wedge B(c_0))$

$(\exists y)\ (A(c_0) \wedge B(y))$

Example of Non-Termination

$(\forall x)\ (\exists y)\ (A(x) \wedge B(y))$

$(\exists y)\ (A(z) \wedge B(y))$ ✓

$(A(z) \wedge B(c_0))$

$(\exists y)\ (A(c_0) \wedge B(y))$ ✓

$(A(c_0) \wedge B(c_1))$

[Each constant introduced by the rule must be new.]

Example of Non-Termination

$(\forall x)\ (\exists y)\ (A(x) \wedge B(y))$

$(\exists y)\ (A(z) \wedge B(y))$ ✓

$(A(z) \wedge B(c_0))$

$(\exists y)\ (A(c_0) \wedge B(y))$ ✓

$(A(c_0) \wedge B(c_1))$

$(\exists y)\ (A(c_1) \wedge B(y))$

Example of Non-Termination

$(\forall x)\ (\exists y)\ (A(x) \wedge B(y))$

$(\exists y)\ (A(z) \wedge B(y))$ ✓

$(A(z) \wedge B(c_0))$

$(\exists y)\ (A(c_0) \wedge B(y))$ ✓

$(A(c_0) \wedge B(c_1))$

$(\exists y)\ (A(c_1) \wedge B(y))$ ✓

$(A(c_1) \wedge B(c_2))$

$(\exists y)\ (A(c_2) \wedge B(y))$

This can go on forever without closing, because the formula is satisfiable.

Using the Tree Method to Find a Model in the Predicate Calculus

- $(\forall x)\,(A(x) \vee B(x)) \rightarrow ((\forall x)A(x) \vee (\forall x)B(x))$
- This formula is not valid, so its negation should be satisfiable.

$\neg((\forall x)\,(A(x) \vee B(x)) \rightarrow ((\forall x)A(x) \vee (\forall x)B(x)))$ ✓

$(\forall x)\,(A(x) \vee B(x))$

$\neg((\forall x)A(x) \vee (\forall x)B(x))$ ✓

$\neg(\forall x)A(x)$ ✓

$\neg(\forall x)B(x)$ ✓

$(\exists x)\,\neg A(x)$

$(\exists x)\,\neg B(x)$

Using the Tree Method to Find a Model in the Predicate Calculus

- So far we have unchecked:

$$(\forall x)\,(A(x) \vee B(x))$$

$$(\exists x)\,\neg A(x)$$

$$(\exists x)\,\neg B(x)$$

Using the Tree Method to Find a Model in the Predicate Calculus

$(\forall x)\ (A(x) \vee B(x))$

$(\exists x)\ \neg A(x)$ ✓

$(\exists x)\ \neg B(x)$ ✓

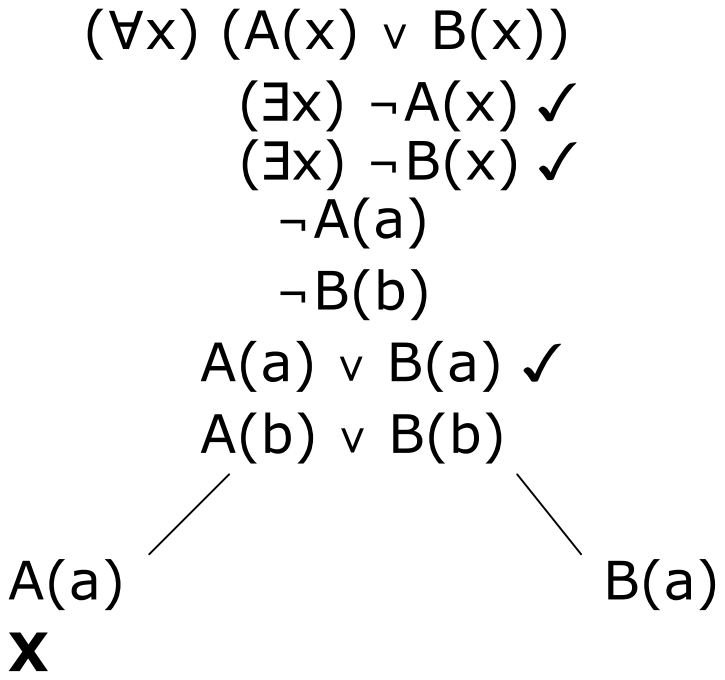
$\neg A(a)$

$\neg B(b)$

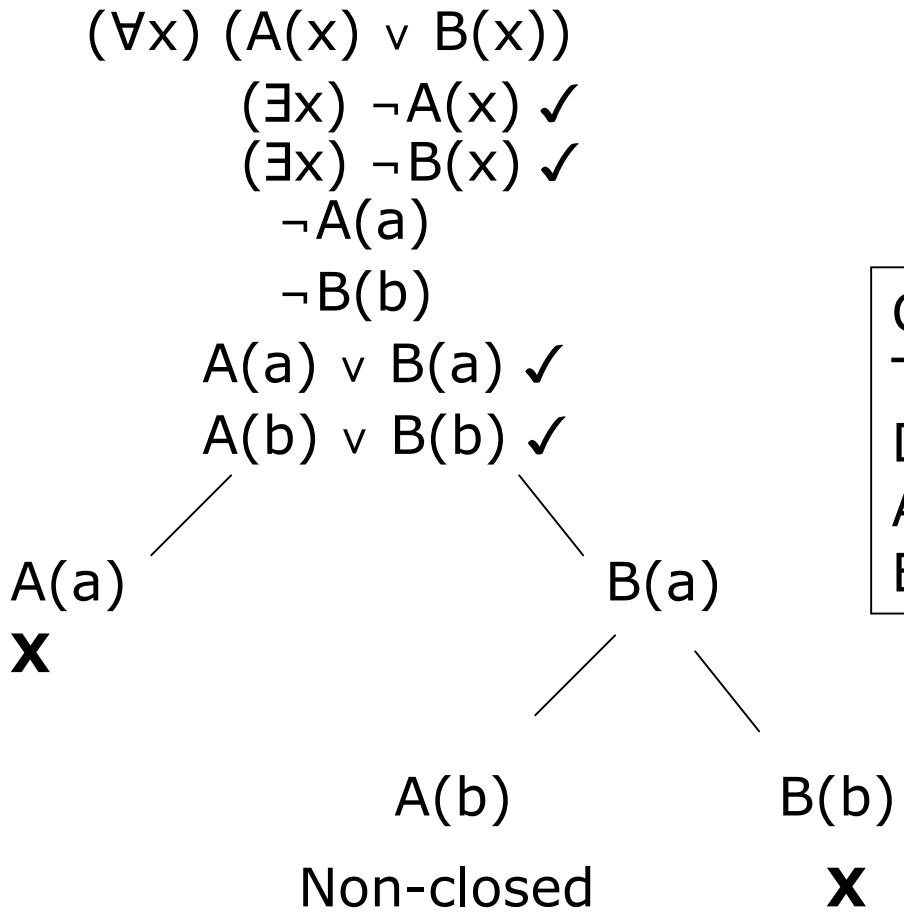
$A(a) \vee B(a)$

$A(b) \vee B(b)$

Using the Tree Method to Find a Model in the Predicate Calculus



Using the Tree Method to Find a Model in the Predicate Calculus



Conclusion:
There is a model
D = {a, b}
A = {(b)}
B = {(a)}