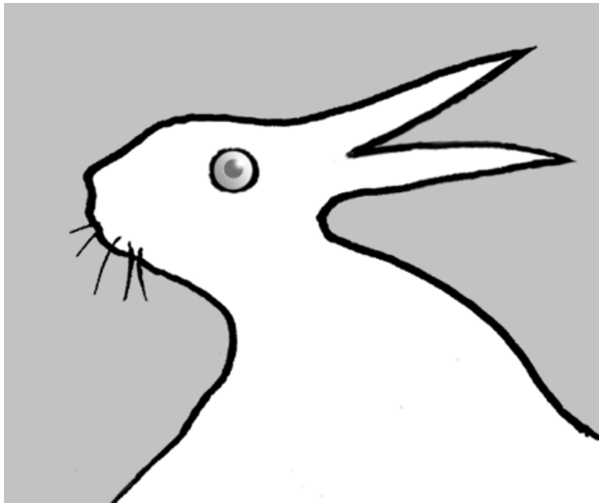
COMPUTATIONAL SEMANTICS: DAY 5

Johan Bos
University of Groningen
www.rug.nl/staff/johan.bos

ESSLLI 2016

Computational Semantics

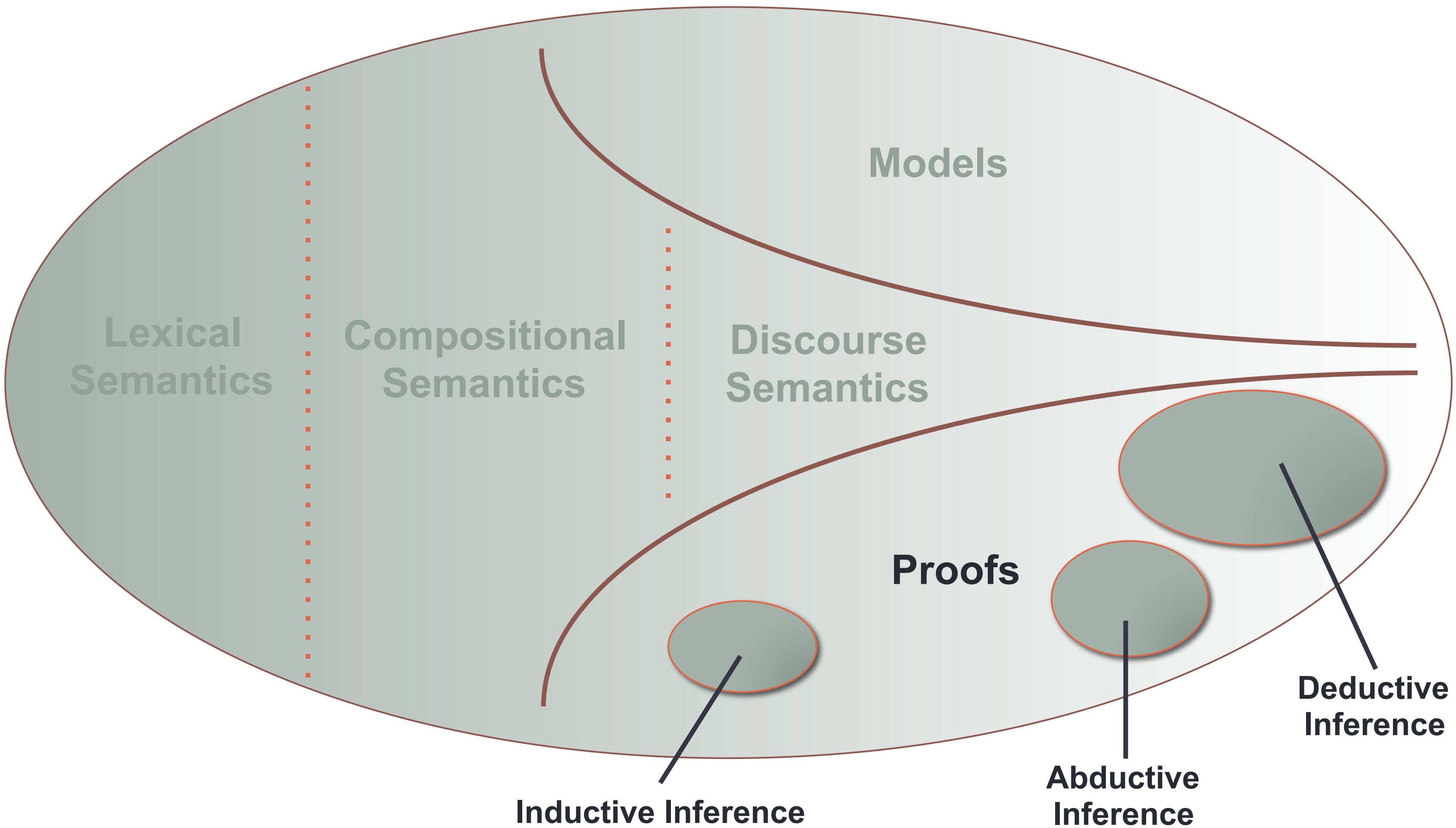
- Day 1: Exploring Models
- Day 2: Meaning Representations
- Day 3: Computing Meanings with DCG
- Day 4: Computing Meanings with CCG
- **Day 5: Drawing Inferences and Meaning Banking**



Drawing Inferences

- By now we know how to produce semantic representations for natural language expressions
- But how can we use them to automate the process of drawing inferences?

Proof-Theoretical Semantics



Abductive reasoning (Abduction)

Guessing for an explanation...

The dog is wet.

? It's raining outside.

? It jumped in the pool.

Inductive reasoning (Induction)

Making generalizations...

This dog has four legs.
That dog has four legs.
And that one. And this one.
And that one too.

All dogs have four legs.

Inductive reasoning (Induction)

Making generalizations...

This dog has four legs.
That dog has four legs.
And that one. And this one.
And that one too.

All dogs have four legs.

Deductive reasoning (Deduction)

Drawing conclusions from a set of premises

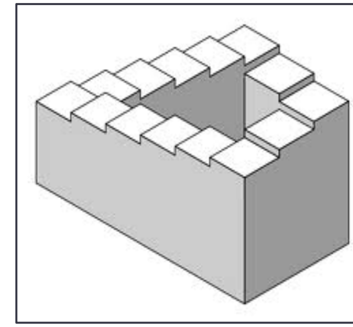
Every dog jumped in the pool.

Fido is a dog.

Fido jumped in the pool.

The three inference tasks

The **Consistency Checking** Task

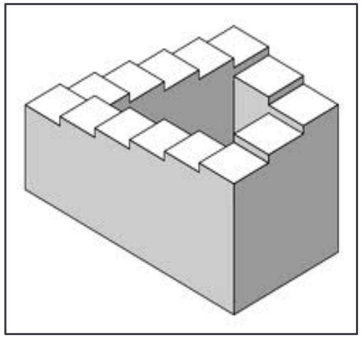


The **Informativeness Checking** Task

The **Querying** Task

The three inference tasks

The Consistency Checking Task
theorem prover + model builder



The Informativeness Checking Task
theorem prover + model builder

The Querying Task
model checker

But hey, isn't first-order logic...

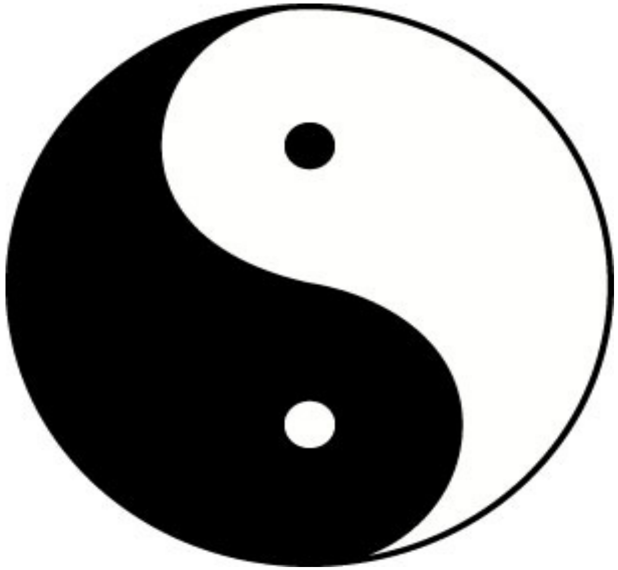
- Yes indeed, first-order logic is undecidable.
  In fact, it is semi-decidable.

- But what does this mean?
  Can we do anything about this?
  Are we in trouble?

No general algorithmic solution

- We already dealt with the querying task (Lecture 1/2)
- The consistency/informativeness checking tasks are undecidable

- But there are partial solutions to be explored:
  - use theorem provers for negative tests
  - use model builders for positive tests



Controlling Inference

expressive power

λx λP higher-order logic

∀P ∃P second-order logic

undecidable

∀x ∃x first-order logic (predicate logic)

Groningen MEANING BANK discourse representation structure

semi-decidable

[] <> modal logics description logics

¬ ∧ → ∨ propositional logic

decidable

Theorem Proving

- The task of checking whether a formula (or a set of formulas) is a validity (a theorem), or put differently, checking whether that formula is true in all models

  Input: **formula**
  Output: **proof** (if you're lucky)

- Theorem proving serves to check whether input is inconsistent and uninformative!

  (i.e., recognizing textual entailment)

Example 1: Steve

Steve visited only Bologna.

Steve visited Bologna and Pisa.

inconsistent

Example 2: Bush

“... when there's more trade, there's more commerce."

not informative

George W. Bush, at the Summit of the Americas in Quebec City, April 21, 2001 (source: Language Log 24/10/2004)

Theorem Proving vs Model Building

- **Theorem provers** check for logical validity
  - Is a formula φ true in all possible situations?
  - Output: proof
  - Useful for: detecting contradictory and non-informative texts

- **Model builders** check for satisfiability
  - Is a formula φ true in at least one situation?
  - Output: model
  - Useful for: detecting consistent and informative texts

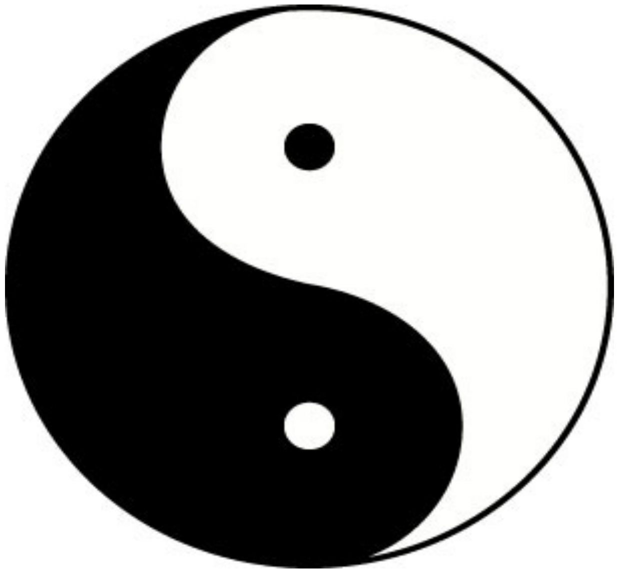
Example 3: James

James visited Rome.

James visited only Rome.

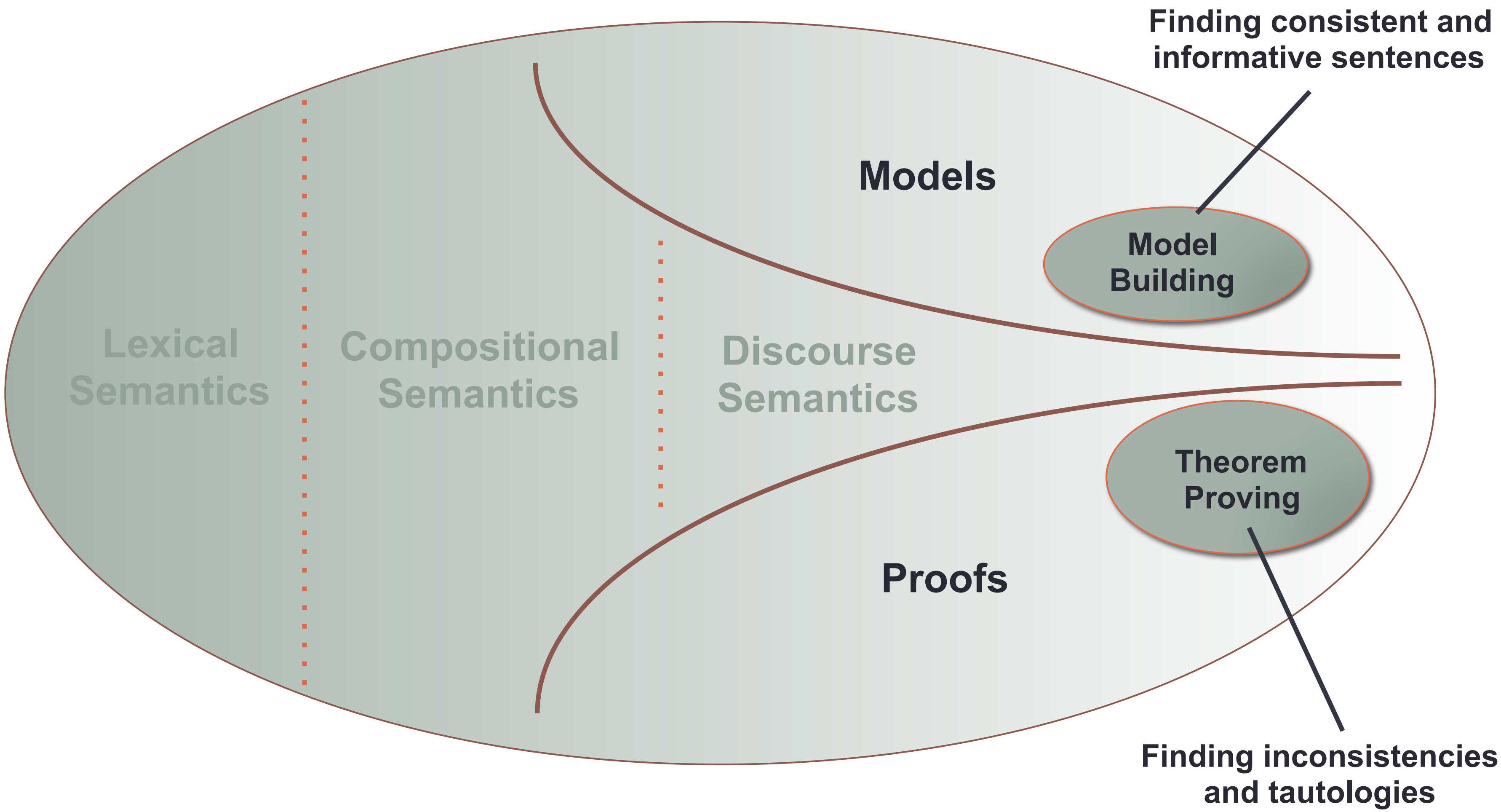
consistent
informative

The Yin and Yang of Inference



Theorem Proving and **Model Building**
function as opposite forces

Inference


Finding consistent and informative sentences
Models
Model Building
Lexical Semantics
Compositional Semantics
Discourse Semantics
Theorem Proving
Proofs
Finding inconsistencies and tautologies

Consistency/Informativeness checking

- ψ is inconsistent wrt $\varphi_1 \ldots \varphi_n$ means that $(\varphi_1 \wedge \ldots \wedge \varphi_n) \rightarrow \neg\psi$ is valid

- ψ is uninformative wrt $\varphi_1 \ldots \varphi_n$ means that $(\varphi_1 \wedge \ldots \wedge \varphi_n) \rightarrow \psi$ is valid

Validity is defined in terms of models:
a valid formula is one that is satisfied in ***all*** models

But there are infinitely many models...

Proof Methods

- Recall the method of truth tables
  - it doesn't scale up
  - and can't be extended to first-order logic
- In this lecture we will look at two specific methods: **semantic tableau** and **resolution**
- We will first look at how this is done for propositional logic. Why?

Because it is a lot simpler than first-order logic!
(dealing with quantifiers and equality is a tricky business)

Propositional Tableaus

- systematic syntactic check for answering the following (semantic) question:

  Suppose we are given a formula and a truth value (true of false). Is is possible to find a model in which the given formula has the given truth value?

- If we had such a systematic check at our disposal, we would be able to test formulas for validity. Why?

  A formula is valid if and only if it is not possible to falsify it in any model

A refutation proof method

- A formula is valid if and only if it is not possible to falsify it in any model
- If tableau can tell us that there is no way to build a model that falsifies a formula, then this formula is valid
- So what we do is this:

We show that a formula is valid by showing that all attempts to falsify it fail

The tableau system

- We will develop **tableau expansion rules**
- They work by breaking down complex formulas into their component formulas
- We will work through three examples. First example:

  p v ¬p

  This is clearly a validity. Why? Let's try to falsify it.

The tableau expansion rules

$T\neg\varphi$
|
$F\varphi$

$F\neg\varphi$
|
$T\varphi$

$T(\varphi \wedge \psi)$
|
$T\varphi$
$T\psi$

$F(\varphi \vee \psi)$
|
$F\varphi$
$F\psi$

$F(\varphi \rightarrow \psi)$
|
$T\varphi$
$F\psi$

$F(\varphi \wedge \psi)$
/ \
$F\varphi$ $F\psi$

$T(\varphi \vee \psi)$
/ \
$T\varphi$ $T\psi$

$T(\varphi \rightarrow \psi)$
/ \
$F\varphi$ $T\psi$

Signed formulas

- We need a nice piece of notation. Here it is:

  Writing Fφ will mean that we want to falsify φ
  Writing Tφ will mean that we want to make φ true

- T and F are called **signs**.
  A formula preceded by a sign is called a **signed formula**.

Proving validity of $p \vee \neg p$

1. F($p \vee \neg p$)

How do you make a disjunction false?

Proving validity of p v ¬p

1. F(p v ¬p) ✓
2. Fp 1, F_v
3. F¬p 1, F_v

This expansion rule is called F_v

The ✓ records the fact that we applied an expansion rule to it (broke it into smaller pieces)

Proving validity of $p \vee \neg p$

1. $F(p \vee \neg p)$ ✓
2. Fp 1, $F_{\vee}$
3. $F\neg p$ 1, $F_{\vee}$ ✓
4. Tp 3, $F_{\neg}$

This expansion rule is called $F_{\neg}$

Proving validity of p v ¬p

1. F(p v ¬p) ✓
2. Fp 1, F_v
3. F¬p 1, F_v ✓
4. Tp 3, $F_\neg$

Two important observations about this tableau:

(1) It is rule-saturated. We can't expand it further.
(2) It is closed. It contains contradictory wishes:
we have to make p false (line 2) and we have to make p true (line 4)

Proving validity of $p \vee \neg p$

1.	$F(p \vee \neg p)$	✓
2.	Fp	1, $F_{\vee}$
3.	$F\neg p$	1, $F_{\vee}$ ✓
4.	Tp	3, $F_{\neg}$

It contains all (just one in this case) possibilities to falsify $p \vee \neg p$. We fail to do this. Hence $p \vee \neg p$ is valid. We call this a closed tableau (or a tableau proof).

Proving validity of $\neg(q \wedge r) \rightarrow (\neg q \vee \neg r)$

1. $F \neg(q \wedge r) \rightarrow (\neg q \vee \neg r)$

Proving validity of $\neg(q \wedge r) \rightarrow (\neg q \vee \neg r)$

1. $F \neg(q \wedge r) \rightarrow (\neg q \vee \neg r)$ ✓
2. $T \neg(q \wedge r)$ 1, $F_{\rightarrow}$
3. $F(\neg q \vee \neg r)$ 1, $F_{\rightarrow}$

Proving validity of ¬(q∧r)→(¬q v ¬r)

1.	F¬(q∧r)→(¬q v ¬r)	✓
2.	T¬(q∧r)	1, $F_{\rightarrow}$
3.	F(¬q v ¬r)	1, $F_{\rightarrow}$ ✓
4.	F ¬q	3, F_{v}
5.	F ¬r	3, F_{v}

Hey! Don't we skip line 2?
No we don't. We're free to apply the rules in any order.

Proving validity of $\neg(q \wedge r) \rightarrow (\neg q \vee \neg r)$

1.	$F\neg(q \wedge r) \rightarrow (\neg q \vee \neg r)$	✓
2.	$T\neg(q \wedge r)$	1, $F_{\rightarrow}$
3.	$F(\neg q \vee \neg r)$	1, $F_{\rightarrow}$ ✓
4.	$F \neg q$	3, $F_{\vee}$ ✓
5.	$F \neg r$	3, $F_{\vee}$
6.	$T\ q$	4, $F_{\neg}$

Proving validity of $\neg(q \wedge r) \to (\neg q \vee \neg r)$

1.	F$\neg(q \wedge r) \to (\neg q \vee \neg r)$	✓
2.	T$\neg(q \wedge r)$	1, $F_{\to}$
3.	F$(\neg q \vee \neg r)$	1, $F_{\to}$ ✓
4.	F $\neg q$	3, F_{v} ✓
5.	F $\neg r$	3, F_{v} ✓
6.	T q	4, $F_{\neg}$
7.	T r	5, $F_{\neg}$

Proving validity of $\neg(q \wedge r) \rightarrow (\neg q \vee \neg r)$

1.	$F \neg(q \wedge r) \rightarrow (\neg q \vee \neg r)$	✓
2.	$T \neg(q \wedge r)$	1, $F_{\rightarrow}$ ✓
3.	$F (\neg q \vee \neg r)$	1, $F_{\rightarrow}$ ✓
4.	$F \neg q$	3, $F_{\vee}$ ✓
5.	$F \neg r$	3, $F_{\vee}$ ✓
6.	$T q$	4, $F_{\neg}$
7.	$T r$	5, $F_{\neg}$
8.	$F q \wedge r$	2, $T_{\neg}$

Proving validity of $\neg(q\wedge r)\rightarrow(\neg q \vee \neg r)$

1.	$F\neg(q\wedge r)\rightarrow(\neg q \vee \neg r)$	✓
2.	$T\neg(q\wedge r)$	1, $F_{\rightarrow}$ ✓
3.	$F(\neg q \vee \neg r)$	1, $F_{\rightarrow}$ ✓
4.	$F\ \neg q$	3, $F_{\vee}$ ✓
5.	$F\ \neg r$	3, $F_{\vee}$ ✓
6.	$T\ q$	4, $F_{\neg}$
7.	$T\ r$	5, $F_{\neg}$
8.	$F\ q\wedge r$	2, $T_{\neg}$ ✓

9. $F\ q$ 8, $F_{\wedge}$

10. $F\ r$ 8, $F_{\wedge}$

Can we further expand this tableau?

1.	$F\neg(q \wedge r) \rightarrow (\neg q \vee \neg r)$	✓
2.	$T\neg(q \wedge r)$	1, $F_{\rightarrow}$ ✓
3.	$F(\neg q \vee \neg r)$	1, $F_{\rightarrow}$ ✓
4.	$F\ \neg q$	3, $F_{\vee}$ ✓
5.	$F\ \neg r$	3, $F_{\vee}$ ✓
6.	$T\ q$	4, $F_{\neg}$
7.	$T\ r$	5, $F_{\neg}$
8.	$F\ q \wedge r$	2, $T_{\neg}$ ✓

9. $F\ q$ 8, $F_{\wedge}$

10. $F\ r$ 8, $F_{\wedge}$

How many branches does this tableau contain?

1.	$F\neg(q \wedge r) \rightarrow (\neg q \vee \neg r)$	✓
2.	$T\neg(q \wedge r)$	$1, F_{\rightarrow}$ ✓
3.	$F(\neg q \vee \neg r)$	$1, F_{\rightarrow}$ ✓
4.	$F\ \neg q$	$3, F_{\vee}$ ✓
5.	$F\ \neg r$	$3, F_{\vee}$ ✓
6.	$T\ q$	$4, F_{\neg}$
7.	$T\ r$	$5, F_{\neg}$
8.	$F\ q \wedge r$	$2, T_{\neg}$ ✓

9. $F\ q$ — $8, F_{\wedge}$

10. $F\ r$ — $8, F_{\wedge}$

Are all branches closed?

Branch 1:
closed (why?)

1.	$F\neg(q \wedge r) \rightarrow (\neg q \vee \neg r)$		✓
2.	$T\neg(q \wedge r)$	$1, F_{\rightarrow}$	✓
3.	$F(\neg q \vee \neg r)$	$1, F_{\rightarrow}$	✓
4.	$F\ \neg q$	$3, F_{\vee}$	✓
5.	$F\ \neg r$	$3, F_{\vee}$	✓
6.	$T\ q$	$4, F_{\neg}$	
7.	$T\ r$	$5, F_{\neg}$	
8.	$F\ q \wedge r$	$2, T_{\neg}$	✓

9.	$F\ q$	$8, F_{\wedge}$	10.	$F\ r$	$8, F_{\wedge}$

Are all branches closed?

1. $F\neg(q \wedge r) \rightarrow (\neg q \vee \neg r)$ ✓
2. $T\neg(q \wedge r)$ — 1, $F_{\rightarrow}$ ✓
3. $F(\neg q \vee \neg r)$ — 1, $F_{\rightarrow}$ ✓
4. $F\ \neg q$ — 3, $F_{\vee}$ ✓
5. $F\ \neg r$ — 3, $F_{\vee}$ ✓
6. $T\ q$ — 4, $F_{\neg}$
7. $T\ r$ — 5, $F_{\neg}$
8. $F\ q \wedge r$ — 2, $T_{\neg}$ ✓

Branch 1: closed (why?)

9. $F\ q$ — 8, $F_{\wedge}$

Branch 2: closed (why?)

10. $F\ r$ — 8, $F_{\wedge}$

Nice so far, but ...

... what happens if the formula we are working which is not a validity?

Checking validity of $(p \wedge q) \rightarrow (r \vee s)$

1. $F (p \wedge q) \rightarrow (r \vee s)$

Checking validity of $(p \wedge q) \rightarrow (r \vee s)$

1.	$F(p \wedge q) \rightarrow (r \vee s)$	✓
2.	$T(p \wedge q)$	1, $F_{\rightarrow}$
3.	$F(r \vee s)$	1, $F_{\rightarrow}$

Checking validity of $(p \wedge q) \rightarrow (r \vee s)$

1. $F(p \wedge q) \rightarrow (r \vee s)$ ✓
2. $T(p \wedge q)$ 1, $F_{\rightarrow}$ ✓
3. $F(r \vee s)$ 1, $F_{\rightarrow}$
4. Tp 2, $T_{\wedge}$
5. Tq 2, $T_{\wedge}$

Checking validity of $(p \wedge q) \rightarrow (r \vee s)$

1. $F(p \wedge q) \rightarrow (r \vee s)$ ✓
2. $T(p \wedge q)$ 1, $F_{\rightarrow}$ ✓
3. $F(r \vee s)$ 1, $F_{\rightarrow}$ ✓
4. Tp 2, $T_{\wedge}$
5. Tq 2, $T_{\wedge}$
6. Fr 3, $F_{\vee}$
7. Fs 3, $F_{\vee}$

Checking validity of $(p \wedge q) \rightarrow (r \vee s)$

1.	$F(p \wedge q) \rightarrow (r \vee s)$	✓
2.	$T(p \wedge q)$	1, $F_{\rightarrow}$ ✓
3.	$F(r \vee s)$	1, $F_{\rightarrow}$ ✓
4.	Tp	2, $T_{\wedge}$
5.	Tq	2, $T_{\wedge}$
6.	Fr	3, $F_{\vee}$
7.	Fs	3, $F_{\vee}$

Can we further expand this tableau?
How many (closed) branches are there?

Checking validity of $(p \wedge q) \rightarrow (r \vee s)$

1.	$F(p \wedge q) \rightarrow (r \vee s)$	✓
2.	$T(p \wedge q)$	1, $F_{\rightarrow}$ ✓
3.	$F(r \vee s)$	1, $F_{\rightarrow}$ ✓
4.	Tp	2, $T_{\wedge}$
5.	Tq	2, $T_{\wedge}$
6.	Fr	3, $F_{\vee}$
7.	Fs	3, $F_{\vee}$

Can we further expand this tableau? NO

How many (closed) branches are there? 1 (open)

Because we are able to falsify the formula, it is not a validity

Checking validity of $(p \wedge q) \rightarrow (r \vee s)$

1.	$F(p \wedge q) \rightarrow (r \vee s)$	✓
2.	$T(p \wedge q)$	1, $F_{\rightarrow}$ ✓
3.	$F(r \vee s)$	1, $F_{\rightarrow}$ ✓
4.	Tp	2, $T_{\wedge}$
5.	Tq	2, $T_{\wedge}$
6.	Fr	3, $F_{\vee}$
7.	Fs	3, $F_{\vee}$

$(p \wedge q) \rightarrow (r \vee s)$ is false in a model
where p is true, q is true, r is false, and s is false

Definitions

- A branch of a tableau is a **closed branch** if it contains both Tφ and Fφ, where φ is some formula
- A branch that is not closed is called an **open branch**
- A tableau with all of its branches closed is called a **closed tableau**
- A tableau with at least one open branch is called an **open tableau**

Semantic Tableaux

- The tableaux method can be used to check for validities (try to falsify a formula, and show that this attempt fails in all possible ways)
- But it can also be used to build a model, i.e. showing that a formula is not a contradiction
- These models can be useful for many applications --- think of our image domain!

In sum: a tableaux system can be used both
as **a theorem prover** and as a **model builder**

Proof Theory & Automated Reasoning

- Investigate logical validity from a purely syntactic perspective
- Various proof methods and theorem provers that implement them
- Crucial:

  only make use of the syntactic structure of formulas

- Examples:
  - **tableau** methods (previous lecture)
  - **resolution** methods (this lecture)

Propositional Resolution

- Introduce a second technique for checking the validity of propositional formulas: the **resolution method**
- It is, like tableau, purely symbolic
- But unlike tableau it uses only one rule (the resolution rule), and needs preprocessing (conversion to CNF).

Conjunctive Normal Form (CNF)

- **positive literals** (sentence symbols: p, q, r, s, ...)
- **negative literals** (negated sentence symbols: ¬p, ¬q, ...)
- **literals**: positive or negative literals
- **clause**: a disjunction of literals
- **CNF**: a conjunction of clauses

Example of a formula in CNF:

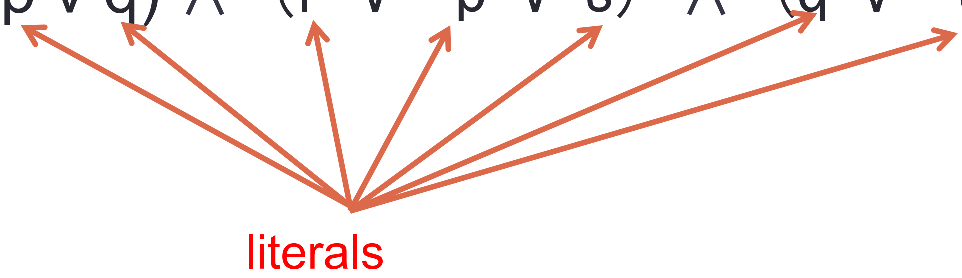
$(p \vee q) \wedge (r \vee \neg p \vee s) \wedge (q \vee \neg s)$

Conjunctive Normal Form (CNF)

- **positive literals** (sentence symbols: p, q, r, s, ...)
- **negative literals** (negated sentence symbols: ¬p, ¬q, ...)
- **literals**: positive or negative literals
- **clause**: a disjunction of literals
- **CNF**: a conjunction of clauses

Example of a formula in CNF:

$(p \lor q) \land (r \lor \neg p \lor s) \land (q \lor \neg s)$



Conjunctive Normal Form (CNF)

- **positive literals** (sentence symbols: p, q, r, s, ...)
- **negative literals** (negated sentence symbols: ¬p, ¬q, ...)
- **literals**: positive or negative literals
- **clause**: a disjunction of literals
- **CNF**: a conjunction of clauses

Example of a formula in CNF:

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

clauses

Key semantic observation: clause

- To make a clause true, we have to make at least one of its literals true (after all, a clause is a disjunction).
- Special case: the empty clause, written as [ ]

  The empty clause contains no literals.
  Hence it is impossible to make at least one of them true.
  Hence it is impossible to make the empty clause true.

Key semantic observation: CNF

- For a formula in CNF to be true, all the clauses it contains (all of the conjuncts) must be true.
- Hence, if a formula in CNF has the empty clause as one of its conjuncts, it can never be true.

Conversion to CNF

- Given an arbitrary formula. How do we get it into CNF?
- One method (there are more):
  - first translate it to negation normal form (NNF)
  - then repeatedly apply the distributive and associative rules
- What is NNF?
  - It is a formula built out of literals, conjunction, and disjunction.

Conversion to NNF

- Rewrite $\neg(\varphi \wedge \psi)$ as $\neg\varphi \vee \neg\psi$
  and $\neg(\varphi \vee \psi)$ as $\neg\varphi \wedge \neg\psi$

drive negations inwards

- Rewrite $\neg(\varphi \rightarrow \psi)$ as $\varphi \wedge \neg\psi$
  and $\varphi \rightarrow \psi$ as $\neg\varphi \vee \psi$

eliminate implications

- Rewrite $\neg\neg\varphi$ as φ

remove double negations

From NNF to CNF

drive conjunction outwards (distribution rules)

- Rewrite $\theta \vee (\varphi \wedge \psi)$ as $(\theta \vee \varphi) \wedge (\theta \vee \psi)$
- Rewrite $(\varphi \wedge \psi) \vee \theta$ as $(\varphi \vee \theta) \wedge (\psi \vee \theta)$

- Rewrite $(\varphi \wedge \psi) \wedge \theta$ as $\varphi \wedge (\psi \wedge \theta)$
- Rewrite $(\varphi \vee \psi) \vee \theta$ as $\varphi \vee (\psi \vee \theta)$

move brackets (associativity rules)

The CNF list of lists notation

(p ∨ q) ∧ (r ∨ ¬p ∨ s) ∧ (q ∨ ¬s)

[[p, q], [r, ¬p, s], [q, ¬s]]

clauses

Set CNF

The resolution algorithm assumes an input formula in **set CNF** (also called *clause sets*):

- None of the clauses may contain a repeated literal
- No clause occurs more than once

Example: [[p,¬q,¬r],[r,q,r]] is not in set CNF. Why?

But [[p,¬q,¬r],[r,q]] is.

(why does this make sense?)

More terminology

- complementary pairs (*resolvents*)
- complementary clauses

Say we have two clauses C and C’. If C contains a positive literal (say r) and C’ its negation (¬r), then C and C’ are **complementary clauses**. Moreover, r and ¬r are a **complementary pair** (are **resolvents**)

The binary resolution rule

- **Input**:
  two complementary clauses
- **Output**:
  one clause obtained by merging the two complementary clauses while removing the resolvents

$$[p_1,\ldots,p_m,\ r\ ,p_{m+1},\ldots,pn] \quad [q1,\ldots,q_i,\ \neg r\ ,q_{i+1},\ldots,q_j]$$

$$[p_1,\ldots,p_m,p_{m+1},\ldots,pn,\ q1,\ldots,q_i,q_{i+1},\ldots,q_j]$$

Why does this make sense?

p v q $\qquad\qquad$ ¬q

$\qquad\qquad$ p

If ¬q is true, then q is false, so to make p v q true, p needs to be true

Why does this make sense?

p v q ¬q v r

----

p v r

It is impossible that both p and r are false (because in that case, either p v q is false, or ¬q v r is false).

Example 1

Proof: (p v ¬p). I.e. try to falsify it.

¬(p v ¬p)
(¬p ∧ ¬¬p)
(¬p ∧ p)

[[p],[¬p]]
[[]]

Empty clause, hence proof.

Example 2

Proof: $\neg(q \wedge r) \rightarrow (\neg q \vee \neg r)$

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

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

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

[-q,-r],[q],[r]

[-r],[r]

[]

Moving to first-order logic

- The tableaux expansion rules are defined for propositional logic. What consequences does moving to FOL have?
  1. We need tableaux expansion rules for the universal and existential quantifier (see Blackburn & Bos chapter 5)
  2. Non-deterministic aspects: the universal quantifier expansion rule can be applied multiple times
  3. Skolem terms for the existential quantifier expansion rules
  4. Unification with occurs-check
  5. Expansion rules for the equality symbol

These directions go beyond the scope of this course. Instead, we will have a look at off-the-shelf model builders

AUTOMATED INFERENCE

Theorem Proving
Model Building

Which theorem provers? Which model builders?

World Cup Automated Deduction (annual event, CASC)

- Best Theorem Provers (**Bliksem**, **Otter**, **Spass, Vampire**)
- Best Model Builders (**Mace**, **Paradox**)

Off-the-shelf model builders

- There are several model builders for first-order logic available (free, easy to install and use)
- In this course we will use the model builder MACE-2, developed by William McCune (1953--2011)

Using the model builder Mace-2

- Downloads: http://www.cs.unm.edu/~mccune/mace2/
  (It comes together with the (famous) theorem prover **Otter**)
- The Blackburn & Bos software contains an interface to mace: it is called `callInference.pl`
- Example query:

```
?- callMB(some(X,and(woman(X),walk(X))),4,Model,Engine).
Model = model([d1],[f(0,c1,d1),f(1,woman,[d1]),f(1,walk,
[d1])]),
Engine = mace.

?- callMB(all(X,imp(woman(X),walk(X))),4,Model,Engine).
Model = model([d1],[f(1,woman,[]),f(1,walk,[])]),
Engine = mace.
```

More about Mace

- Mace builds **finite** models
- There are models that are **infinitely** large; so model builders such as mace try to build a model up to a given domain size (the second argument of callMB/4)
- Model builders (obviously) don't know anything about the world!

```
  ?- callMB(some(X,and(man(X),woman(X))),4,Model,Engine).
Model = model([d1],[f(0,c1,d1),f(1,man,[d1]),f(1,woman,[d1])]),
Engine = mace.
```

Reflection

- What can we use theorem provers for?
- What can we use model builders for?
- Other uses of the model checker?

General Purpose – Specific Applications

Logicians vs. Linguists

Suppose we got a theory Φ

	Logician	Linguist
Proof Φ		
Model Φ		
Proof ¬Φ		
Model ¬Φ		

Logicians vs. Linguists

Suppose we got a theory Φ

	Logician	Linguist
Proof Φ	☺	
Model Φ		
Proof ¬Φ		
Model ¬Φ		

Logicians vs. Linguists

Suppose we got a theory Φ

	Logician	**Linguist**
Proof Φ	☺	☹
Model Φ		
Proof ¬Φ		
Model ¬Φ		

Logicians vs. Linguists

Suppose we got a theory Φ

	Logician	Linguist
Proof Φ	☺	☹
Model Φ	☹	
Proof ¬Φ		
Model ¬Φ		

Logicians vs. Linguists

Suppose we got a theory Φ

	Logician	Linguist
Proof Φ	☺	☹
Model Φ	☹	☺
Proof ¬Φ		
Model ¬Φ		

Logicians vs. Linguists

Suppose we got a theory Φ

	Logician	Linguist
Proof Φ	☺	☹
Model Φ	☹	☺
Proof ¬Φ	☺	☹
Model ¬Φ	☹	☺

Logician vs. Linguists

Summing up:

- The logician thinks in terms of **proofs** and **counter-models**
- The linguist thinks in terms of **models** and **counter-proofs**