Artificial Intelligence

6. Automated Reasoning

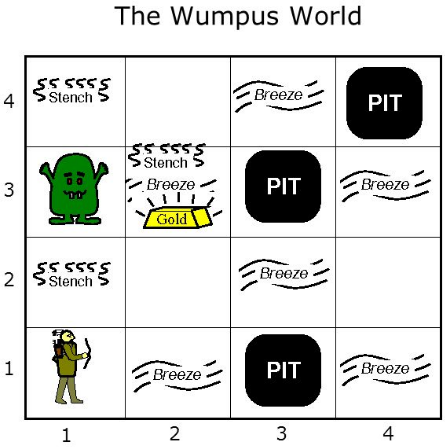
Prof. Bojana Dalbelo Bašić
Assoc. Prof. Jan Šnajder

University of Zagreb
Faculty of Electrical Engineering and Computing

Academic Year 2019/2020

2.3

Motivation: The world of Wumpus



Percepts (facts):

$\neg B_{1,1}$
$\neg B_{1,2}$
$B_{2,1}$
$\neg P_{1,1}$

Knowledge (rules):

$B_{2,1} \leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$

New knowledge (conclusions):

$P_{2,2} \vee P_{3,1}$

Example: Customs officers and diplomats

Premises

Customs officers searched everyone who entered the country but wasn't a diplomat. Some smugglers who entered the country were searched only by smugglers. No smuggler is a diplomat.

Conclusion

Some customs officers are smugglers.

Conclusion = logical consequence?

- The conclusion (newly derived knowledge) is nothing but the **logical consequence** of the premises (existing knowledge)
- Thus, deriving new knowledge amounts to proving logical consequences of existing knowledge

$$\text{existing knowledge} \vDash \text{new knowledge}$$

- However, in practice we run into problems...

Problems with proving semantic consequences

(1) **Intractability**:

- We need to check 2^n interpretations
- Imagine we wish to prove $F_1, \ldots, F_{100} \vDash F_1$. Assuming also $n = 100$, we should check 1.27×10^{30} interpretations. This is impossible!
- Moreover, in this particular case it seems an overkill because it is *evident* that the semantic consequence relation holds

(2) **Undecidability**:

- In FOL, the number of interpretations is infinite, thus we have no chance to check all interpretations
- Actually, FOL is **semi-decidable**: we can prove validity if it holds, but if it doesn't, we cannot always prove that it doesn't

- Is there a way out of these problems? Yes, to some extent
- Instead to deal with interpretations and models (i.e., semantics), we should shift to using rules of inference (i.e, proof theory)

Outline

Outline

1. Proof theory
2. Resolution in propositional logic (PL)
3. Refutation resolution in propositional logic (PL)
4. Resolution in first-order logic (FOL) – prep
5. Resolution in first-order logic (FOL) – method and examples

Proof theory

- People don't really draw the conclusions by proving semantic consequences (it's not that they try to find interpretations that satisfy F, but don't satisfy G)!
- Instead of this, they try to show that G can be **derived** from the premises using a number of **inference rules**
- Each rule needs to be *justified* and simple
- Rules of inference enable us to derive new formulas from given premises, without making any explicit reference to the semantics of logic (the truth values of propositional variables)

Proof theory

- Thus, proof theory offers two benefits over proving semantic consequences:
  - **efficiency:** instead of searching exhaustively through all the interpretations, we can prove consequences faster (especially when using a smart proof strategy). Note, however, that we cannot escape undecidability of FOL
  - **interpretability:** we can explain why something follows from the premises (by appealing to the rules of inference). We get a *proof*

Deductive consequence

Deductive consequence

Formula G is a **deduction** or **deductive consequence** of formulas $F_1, F_2, \ldots, F_n$ if and only if G can be **derived** from the premises $F_1, F_2, \ldots, F_n$ using rules of inference.

We write $F_1, F_2, \ldots, F_n \vdash G$ and read "$F_1, \ldots, F_n$ **derives** or **deductively entails** G".

Theorem

Formula G is a **theorem** if and only if $\vdash G$ holds, i.e., formula G can be derived from an empty set of premises.

- Proving $F \vdash G$ is equivalent to proving $\vdash F \rightarrow G$
- This is why we talk of **theorem proving** instead of deriving deductions

Rules of inference

- An example of an inference rule:
  "If two propositions are true, then their conjunction is true as well"

Conjunction introduction rule

$$\frac{A \qquad B}{A \wedge B} \qquad \text{or} \qquad A, B \vdash A \wedge B$$

- What about the following rule?

$$A \vee B \vdash A$$

- Intuitively, the first rule is **semantically correct**, whereas the second rule isn't
- For a rule to be correct, its deductive consequence must also be the **semantic consequence** of the premises

Soundness and completeness

Soundness

An inference rule is **sound** if, when applied to a set of premises, derives a formula that is a **logical consequence** of these premises.

Formally, a rule of inference r is sound if and only if

$$\text{if } F_1, \ldots, F_n \vdash_r G \text{ then } F_1, \ldots, F_n \vDash G$$

Completeness

A set of rules R is **complete** if and only if it can be used to derive all logical consequences:

$$\text{if } F_1, \ldots, F_n \vDash G \text{ then } F_1, \ldots, F_n \vdash_R G$$

Soundness and completeness tie together semantics and proof theory (they provide a two-way link between $\vdash$ and $\vDash$)

Soundness and completeness – an example

- Let's prove that $F \rightarrow G, F \ \vdash \ G$ **(modus ponens)** is a sound rule. We must prove $F \rightarrow G, F \ \vDash \ G$. Direct proof:

F	G	$F \rightarrow G$	$(F \rightarrow G) \wedge F$	$\big((F \rightarrow G) \wedge F\big) \rightarrow G$
$\perp$	$\perp$	$\top$	$\perp$	$\top$
$\perp$	$\top$	$\top$	$\perp$	$\top$
$\top$	$\perp$	$\perp$	$\perp$	$\top$
$\top$	$\top$	$\top$	$\top$	$\top$

- Let's prove that $F \rightarrow G, G \vdash F$ **(abduction)** is not sound. We must prove $F \rightarrow G, G \ \nvDash \ F$. Direct proof:

F	G	$F \rightarrow G$	$(F \rightarrow G) \wedge G$	$\big((F \rightarrow G) \wedge G\big) \rightarrow F$
$\perp$	$\perp$	$\top$	$\perp$	$\top$
$\perp$	$\top$	$\top$	$\top$	$\perp$
$\top$	$\perp$	$\perp$	$\perp$	$\top$
$\top$	$\top$	$\top$	$\top$	$\top$

Proving theorems

- In AI, we are interested in doing theorem proving automatically
- This is what **automated reasoning** and **automated theorem proving (ATP)** are about
- A tool that implements a proof method is called a **theorem prover**
- What theorem provers derive must, of course, be **semantically correct**, i.e., justifiable in semantic terms

Proof methods

- Proof theory has provided us with many **proof methods**
- The method must be semantically sound, and preferably also complete
    - natural deduction systems
    - axiomatic (Hilbert-style) systems
    - sequent calculi
    - tableau method
    - resolution method
- We will focus on the resolution method. This method is sound and complete, and can easily be implemented on a computer

Outline

Resolution method

- **Resolution method** is used in propositional logic and first-order predicate logic
- First proposed by J. A. Robinson in 1965
- The method consists of a single inference rule:

Resolution rule

$$\frac{A \vee F \qquad \neg A \vee G}{F \vee G} \qquad \text{or} \qquad A \vee F,\ \neg A \vee G \ \vdash\ F \vee G$$

which is equivalent to:

$$\neg F \to A,\ A \to G \ \vdash\ \neg F \to G$$

- The advantage here is that we're working with a single rule, which greatly simplifies automated reasoning

Resolution rule – soundness

- Let's convince ourselves that the resolution rule is **sound**
- We must prove that the deductive consequence derived with the resolution rule also is a semantic consequence. I.e., we must prove:

$$A \vee F,\ \neg A \vee G \ \vDash\ F \vee G$$

- For example, using the direct method:

A	F	G	$\overbrace{A \vee F}^{P}$	$\neg A$	$\overbrace{\neg A \vee G}^{Q}$	$P \wedge Q$	$\overbrace{F \vee G}^{R}$	$(P \wedge Q) \rightarrow R$
$\top$	$\top$	$\top$	$\top$	$\bot$	$\top$	$\top$	$\top$	$\top$
$\top$	$\top$	$\bot$	$\top$	$\bot$	$\bot$	$\bot$	$\top$	$\top$
$\top$	$\bot$	$\top$	$\top$	$\bot$	$\top$	$\top$	$\top$	$\top$
$\top$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\bot$	$\bot$	$\top$
$\bot$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
$\bot$	$\top$	$\bot$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
$\bot$	$\bot$	$\top$	$\bot$	$\top$	$\top$	$\bot$	$\top$	$\top$
$\bot$	$\bot$	$\bot$	$\bot$	$\top$	$\top$	$\bot$	$\bot$	$\top$

Clause

- Resolution rule can only be applied to premises that are disjunctions
- If we wish to be able to apply the resolution rule many times over, all premises must be in a form of disjunctions of literals. This form is called a **clause**

Clause

A **literal** is an atom or its negation. A **clause** is a disjunction of finitely many literals G_i:

$$G_1 \vee G_2 \vee \cdots \vee G_n, \quad n \geq 0$$

A clause containing a single literal is called a **unit clause**.

- Examples of literals: A, F, $\neg A$, $\neg F$, G, $\neg G$
- Examples of clauses: $A \vee F$, $\neg A \vee G$, $A \vee \neg B \vee C \vee \neg D$, F

Resolution on clauses

Resolution rule on PL clauses

$$\frac{F_1 \vee \cdots \vee F_i \vee \cdots \vee F_n \qquad\qquad G_1 \vee \cdots \vee G_i \vee \cdots \vee G_m}{F_1 \vee \cdots \vee F_{i-1} \vee F_{i+1} \vee \cdots \vee F_n \vee G_1 \vee \cdots \vee G_{i-1} \vee G_{i+1} \vee \cdots \vee G_m}$$

where F_i and G_i are **complementary literals** (one is the negation of the other). The premises are called the **input clauses** and the deduction is called the **resolvent**.

- Examples:
  $A \vee B \vee \neg C,\ D \vee \neg B \vee E \ \vdash\ A \vee \neg C \vee D \vee E$
  $\neg A \vee B,\ A \ \vdash\ B$
  $A \vee B,\ A \vee \neg B \ \vdash\ A \vee A$
  $A \vee B,\ \neg A \vee \neg B \ \vdash\ B \vee \neg B$
  $A \vee B,\ \neg A \vee \neg B \ \vdash\ A \vee \neg A$

Conjunctive normal form

- If the premises are clauses, the set of premises is a **conjunction of clauses** (the premises are implicitly conjoined with $\wedge$)
- **Q:** Does this restrict the application of resolution? **A:** No!
- Every formula can be represented as a conjunction of clauses by conversion into the **conjunctive normal form**

Conjunctive normal form (CNF)

Formula F is in a **conjunctive normal form** iff F is in the form

$$F_1 \wedge F_2 \wedge \cdots \wedge F_n$$

where F_i is in the form

$$G_{i1} \vee G_{i2} \vee \cdots \vee G_{im}$$

where G_{ij} are literals (atoms or their negations).

Conversion into CNF

- Every formula can be converted into CNF in four sequential steps:

Conversion into CNF

(1) Equivalence elimination: $F \leftrightarrow G \;\equiv\; (\neg F \vee G) \wedge (\neg G \vee F)$

(2) Implication elimination: $F \rightarrow G \;\equiv\; \neg F \vee G$

(3) Moving negations onto atoms: $\neg(F \vee G) \;\equiv\; \neg F \wedge \neg G$
$\neg(F \wedge G) \;\equiv\; \neg F \vee \neg G$

(4) Applying distributive law: $F \vee (G \wedge H) \;\equiv\; (F \vee G) \wedge (F \vee H)$

Each step is applied repeatedly until the result no longer changes.
Involution $\neg\neg F \equiv F$ is applied in each step whenever possible.

Conversion into CNF – an example

$$(C \vee D) \rightarrow (\neg A \leftrightarrow B)$$

(1) Eliminate equivalences:

$$\begin{aligned}
&(C \vee D) \rightarrow (\neg A \leftrightarrow B)\\
&(C \vee D) \rightarrow \big((\neg\neg A \vee B) \wedge (\neg B \vee \neg A)\big)
\end{aligned}$$

(2) Eliminate implications:

$$(C \vee D) \rightarrow \big((A \vee B) \wedge (\neg B \vee \neg A)\big)$$

(3) Move negations in:

$$\neg(C \vee D) \vee \big((A \vee B) \wedge (\neg B \vee \neg A)\big)$$

(4) Apply distributive laws:

$$\begin{aligned}
&(\neg C \wedge \neg D) \vee \big((A \vee B) \wedge (\neg B \vee \neg A)\big)\\
&\big((\neg C \wedge \neg D) \vee (A \vee B)\big) \wedge \big((\neg C \wedge \neg D) \vee (\neg B \vee \neg A)\big)\\
&\big((\neg C \vee A \vee B) \wedge (\neg D \vee A \vee B)\big) \wedge \big((\neg C \wedge \neg D) \vee (\neg B \vee \neg A)\big)\\
&\big((\neg C \vee A \vee B) \wedge (\neg D \vee A \vee B)\big) \wedge \big((\neg C \vee \neg B \vee \neg A) \wedge (\neg D \vee \neg B \vee \neg A)\big)\\
&(\neg C \vee A \vee B) \wedge (\neg D \vee A \vee B) \wedge (\neg C \vee \neg B \vee \neg A) \wedge (\neg D \vee \neg B \vee \neg A)
\end{aligned}$$

Clausal form

- A CNF formula can be represented as a set of clauses implicitly conjoined with $\wedge$
- Each clause can be represented as set of literals implicitly conjoined with $\vee$
- Thus, a formula can be represented as a **set of sets of literals**
- This is called the **clausal form**
- For example:
  $(\neg C \vee A \vee B) \wedge (\neg D \vee A \vee B) \wedge (\neg C \vee \neg B) \;\Rightarrow$
  $\big\{\{\neg C, A, B\}, \{\neg D, A, B\}, \{\neg C, \neg B\}\big\}$
- We also often write the clauses one below the other:
  $\neg C \vee A \vee B$
  $\neg D \vee A \vee B$
  $\neg C \vee \neg B$

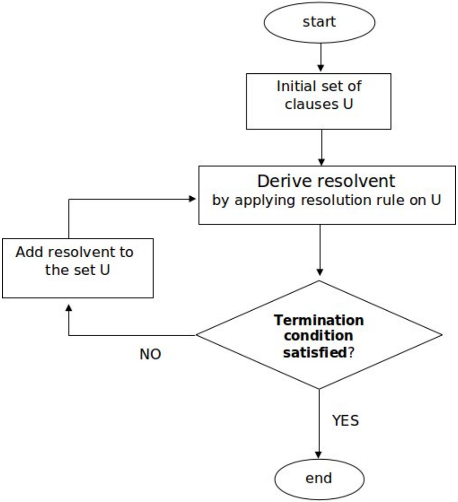
Resolution

The inference step is repeated until:

(1) the goal formula is derived

(2) no further formulae can be derived

(3) computational resources are exhausted



Resolution – an example

- Let us use resolution to prove: $A \rightarrow B, B \rightarrow C, A \ \vdash \ C$
- Premises in clausal form:
  (1) $\neg A \vee B$
  (2) $\neg B \vee C$
  (3) A
- Derivation:
  (4) $\neg A \vee C$ (from 1 & 2)
  (5) C (from 3 & 4)
- Alternative derivation:
  (4') B (from 1 & 3)
  (5') C (from 2 & 4')

Incompleteness of resolution

- We've proved that that the resolution rule is sound. But is it complete?
- We can easily show that the resolution rule is not complete
- E.g., let us consider the deduction $F \vdash F \vee G$
- We cannot derive this using the resolution rule (why not?)
- But $F \vDash F \vee G$ does hold (check this!)
- Since $F \vDash F \vee G$ holds, but we cannot derive $F \vdash F \vee G$ using the resolution rule, it follows that the resolution rule cannot derive all logical consequences, and hence the **resolution rule is not complete**

Outline

Direct vs. refutation resolution

- The resolution procedure we've been using thus far is the **direct resolution**, which tries to derive G from $F_1, \ldots, F_n$
- There also exists **refutation resolution**, which instead tries to prove that $F_1 \wedge \cdots \wedge F_n \wedge \neg G$ is inconsistent
- Direct resolution is **incomplete**, but refutation resolution is **complete**

Refutation resolution (1)

- Instead of trying to prove $F_1, \dots, F_n \vdash G$, we try to prove that $F_1 \wedge \dots \wedge F_n \wedge \neg G$ is inconsistent
- As a special case of the resolution rule we have:

$$\frac{A \qquad \neg A}{\mathrm{NIL}}$$

- NIL denotes the empty clause whose semantic value is $\perp$
- If the resolution procedure derives NIL, then this means that the premises are inconsistent (because the resolution rule is sound)

Refutation resolution (2)

- It has been proven that, whenever the set of clauses is inconsistent, resolution will derive the NIL clause (**ground resolution theorem**)
- Thus we can always prove the inconsistency of a set of clauses
- This also means that we can prove all logical consequences. **Q:** Why?
- **A:** Because we can prove $F \models G$ using the refutation method, by proving that $F \wedge \neg G$ is inconsistent
- This then means that refutation resolution is complete, because we can use it prove all logical consequences
- Therefore, refutation resolution is both **sound and complete**!

Refutation resolution – example 1

- Let us show that we can use refutation resolution to prove $F \ \vdash \ F \vee G$:
- Negation of the goal formula: $\neg(F \vee G) \equiv \neg F \wedge \neg G$
- Set of clauses:

  (1) F
  (2) $\neg F$
  (3) $\neg G$

- From (1) and (2) we derive NIL
- This proves that the set of premises is inconsistent, i.e., that $F \vee G$ is the deductive/logical consequence of F

Refutation resolution – example 2

A diplomatic problem

As a Protocol Representative, you're in charge of sending out invitations for the Annual Diplomats Ball. You are to take into account the following:

(1) The ambassador wants you to also invite UK, if you invite Turkey.

(2) The vice-ambassador wants you to invite Turkey or Argentina.

(3) Due to a recent diplomatic incident, you cannot invite both UK and Argentina.

Whom to invite?

Let's prove: *"If I invite Turkey, I will not invite Argentina"*

- Logical representation of the problem:
  $(T \to U) \land (T \lor A) \land \neg(U \land A)$
- We need to prove $T \to \neg A$ (the goal)

Refutation resolution – example 2

- Conversion into clauses:

  (1) $\neg T \vee U$
  (2) $T \vee A$
  (3) $\neg U \vee \neg A$

- Negation of the goal: $\neg(T \rightarrow \neg A) \equiv \neg(\neg T \vee \neg A) \equiv T \wedge A$
- Adding new clauses:

  (4) T
  (5) A

- Resolution procedure:

  (6) U (from 1 & 4)
  (7) $\neg A$ (from 3 & 6)
  (8) NIL (from 5 & 7)

Factorization

- Refutation resolution is complete, provided the clauses are **factorized**
- Factorization uses equivalence $G \vee G \equiv G$ to eliminate multiple occurrences of a literal in a formula
- For example:
  $\neg A \vee \neg A,\ A \vee A\ \vdash\ A \vee \neg A$
- The set of clauses is inconsistent, but we've derived a valid formula
- **Q:** Is this sound? **A:** Of course it is. A valid formula is a logical consequence of any formula. But this is not very useful.
- However, if we first factorize, we get:
  $\neg A,\ A\ \vdash\ \mathrm{NIL}$
- To retain completeness, we must apply factorization whenever possible

Refutation resolution algorithm

Refutation resolution algorithm (for propositional logic)

function $\mathrm{plResolution}(F, G)$
  $clauses \leftarrow \mathrm{cnfConvert}(F \wedge \neg G)$
  $new \leftarrow \emptyset$
  loop do
    for each (c_1, c_2) **in** $\mathrm{selectClauses}(clauses)$ **do**
      $resolvents \leftarrow \mathrm{plResolve}(c_1, c_2)$
      if $\mathrm{NIL} \in resolvents$ **then return** $true$
      $new \leftarrow new \cup resolvents$
    if $new \subseteq clauses$ **then return** $false$
    $clauses \leftarrow clauses \cup new$

- $\mathrm{cnfConvert}$ – converts a formula into a CNF
- $\mathrm{selectClauses}$ – selects a set of clause pairs to resolve
- $\mathrm{plResolve}$ – resolves two clauses and returns a set of resolvents

Refutation resolution algorithm – remarks

- Applying the resolution rule on a pair of clauses may result in more than one resolvent, hence the algorithm uses a set of resolvents
- To retain completeness, **factorization** should be applied on each derived resolvent
- The number of possible distinct clauses is finite (when factorized), thus the algorithm certainly terminates in a finite number of steps
- No pair of clauses should be resolved more than once
- Deriving a NIL clause from a set of clauses is a **search problem**: in each step we must choose which pair of clauses to resolve
- So we need a **search strategy**, which in the context of resolution is called a **resolution strategy**

Resolution strategies

- There are two types of **resolution strategies**:
  - simplification strategies
  - control strategies
- **Simplification strategies** are used to delete redundant or irrelevant clauses generated during the proof procedure, avoiding unnecessary steps later on
- **Control strategies** determine the order in which clauses are resolved
- The strategy needs to be **complete**: it must derive NIL when given a set of inconsistent clauses
- This is not to be confused with completeness of inference rules (in general, for a proof procedure to be complete, it must combine complete inference rules with a complete search strategy)

Simplification strategy

Deletion strategy

Removal of **redundant** clauses:

- A clause that is **subsumed** by another clause may be deleted
- Based on the absorption equivalence: $F \wedge (F \vee G) \equiv F$
- If the set contains a pair of clauses C_1 and C_2 such that $C_1 \subseteq C_2$, then clause C_2 may be removed from the set (clauses are represented as sets of literals)

Removal of **irrelevant** clauses:

- Every **valid** clause (tautology) is irrelevant (why?)
- If the resolvent is a valid clause, it may be removed immediately
- Checking the validity of a clause is simple: a clause is valid iff it contains a complementary pair of literals F_i and $\neg F_i$

Control resolution strategies

Level saturation strategy

- Resolvents are derived level-by-level (as in breadth-first search): we resolve all pairs of first-level clauses (initial set of clauses), then we resolve second-level clauses, etc.
- At i-th level, the input clauses come from level 1 down to level $(i-1)$
- This strategy is complete, but very inefficient (the problem of combinatorial explosion)

Control resolution strategies

Set-of-support strategy (SoS)

- Builds on the assumption that the set of input premises is **consistent**
- For if it were not consistent, it would logically entail any formula!
- Thus, to prove inconsistency in refutation resolution, we have to combine the input premises' clauses with clauses from the negated goal, or with the newly derived clauses
- **Set of support (SoS)**: the clauses obtained from the negated goal as well as all subsequently derived clauses
- **Set-of-support strategy**: at least one parent clause always comes from the SoS
- SoS increases as we derive more clauses
- SoS strategy is complete and in principle more efficient than level saturation strategy (especially if the SoS is small)

Outline

Example: Customs officers and diplomats

Premises

Customs officers searched everyone who entered the country but wasn't a diplomat. Some smugglers who entered the country were searched only by smugglers. No smuggler is a diplomat.

Conclusion

Some customs officers are smugglers.

A simpler example

(1) *Every student attends lectures.*
(2) *John is a student.*
$\vdash$ *John attends the lectures.*

Inference using **natural deduction**:

(1) $\forall x\big(S(x) \rightarrow L(x)\big)$
(2) $S(John)$
(3) $S(John) \rightarrow L(John)$ (from 1 by rule of universal instantiation)
(4) $L(John)$ (from 2 and 3 by rule of modus ponens)

Universal instantiation + modus ponens = generalized modus ponens

How can we derive this using resolution rule?

Resolution in FOL – a sketch

1. Convert the formulas (premises and negated goal) into clausal form:

$$
\begin{array}{lcl}
\forall x\big(S(x) \to L(x)\big) & \Rightarrow & \neg S(x) \vee L(x) \\
S(John) & \Rightarrow & S(John) \\
L(John) & \Rightarrow & \neg L(John)
\end{array}
$$

2. Match complementary literals:

$$
\begin{array}{rcll}
\neg S(x) & \Leftrightarrow & S(John) & \text{if } x \leftarrow John \\
L(x) & \Leftrightarrow & L(John) & \text{if } x \leftarrow John
\end{array}
$$

$\Rightarrow$ operation of unification, resulting in variable substitution

3. Resolve clauses according to the obtained substitution:

$$
\begin{array}{lcl}
\neg S(x) \vee L(x), S(John) & \vdash & L(John) \\
L(John), \neg L(John) & \vdash & \mathrm{NIL}
\end{array}
$$

Conversion into clausal form

- As in PL, resolution in FOL requires the formula to be in the **clausal form**
- It is implicitly assumed that:
  - all variables are **universally quantified**
  - clauses are conjoined into disjunctions
- Furthermore, all FOL clauses need to be **standardized** –
- Conversion into clausal form is done in **10 sequential steps**

Conversion into clausal form

- **Step 1: Equivalence elimination**

$$F \leftrightarrow G \equiv (\neg F \lor G) \land (\neg G \lor F)$$

- **Step 2: Implication elimination**

$$F \rightarrow G \equiv \neg F \lor G$$

- **Step 3: Pushing negations onto atoms**

$$\begin{aligned}
\neg(F \lor G) &\equiv \neg F \land \neg G \\
\neg(F \land G) &\equiv \neg F \lor \neg G \\
\neg \forall x F(x) &\equiv \exists x(\neg F(x)) \\
\neg \exists x F(x) &\equiv \forall x(\neg F(x))
\end{aligned}$$

- In each of the steps above, whenever applicable use **involution** $\neg\neg F \equiv F$ to eliminate double negation

Conversion into clausal form

- **Step 4: Variable renaming** so that every quantifier binds a unique variable

$$(\forall x F(x) \lor \forall x G(x)) \equiv (\forall x F(x) \lor \forall y G(y))$$
$$(\forall x F(x) \lor \exists x G(x)) \equiv (\forall x F(x) \lor \exists y G(y))$$
$$(\exists x F(x) \lor \forall x G(x)) \equiv (\exists x F(x) \lor \forall y G(y))$$
$$(\exists x F(x) \lor \exists x G(x)) \equiv (\exists x F(x) \lor \exists y G(y))$$

$$(\forall x F(x) \land \forall x G(x)) \equiv (\forall x F(x) \land \forall y G(y))$$
$$(\forall x F(x) \land \exists x G(x)) \equiv (\forall x F(x) \land \exists y G(y))$$
$$(\exists x F(x) \land \forall x G(x)) \equiv (\exists x F(x) \land \forall y G(y))$$
$$(\exists x F(x) \land \exists x G(x)) \equiv (\exists x F(x) \land \exists y G(y))$$

Conversion into clausal form – skolemization

- **Step 5: Skolemization** – replacement of all existentially quantified variables with Skolem expressions
- If the existentially quantified variable does not depend on other variables: replace with a Skolem constant

$$
\begin{array}{rl}
& \exists x \mathrm{SISTER}(x, John) \\
\Rightarrow & \mathrm{SISTER}(\underbrace{Anna}_{\text{Skolem constant}}, John)
\end{array}
$$

- If the existentially quantified variable depends on universally quantified variables: replace with a Skolem function

$$
\begin{array}{rl}
& \forall x \exists y \mathrm{MOTHER}(y, x) \\
\Rightarrow & \mathrm{MOTHER}(\underbrace{f(x)}_{\text{Skolem function}}, x)
\end{array}
$$

Conversion into clausal form – skolemization

- The arguments of a Skolem function are all those universally quantified variables whose scope includes the existentially quantified variables being replaced

$$\begin{aligned} & \exists u \forall v \forall w \exists x \forall y \exists z F(u, v, w, x, y, z) \\ \Rightarrow \quad & \forall v \forall w \forall y F(a, v, w, f(v, w), y, g(v, w, y)) \end{aligned}$$

None of the symbols a, f, and g must occur in the original formula

Thoraf Albert Skolem (1887–1963)

Norwegian mathematician and logician, one of the founders of model theory.

Conversion into clausal form – skolemization

- How is skolemization justified? What are we allowed to replace an existentially quantified variable with an arbitrary constant?

$$\exists x \mathrm{SISTER}(x, John) \overset{???}{\equiv} \mathrm{SISTER}(Anna, John)$$

- The above equivalence generally doesn't hold, but that's not the point
- The point is that **skolemization does not affect the satisfiability of a formula!**

$$\begin{array}{rl} \text{If} & \exists x \mathrm{SISTER}(x, John) \equiv \bot \\ \text{then} & \mathrm{SISTER}(Anna, John) \equiv \bot \end{array}$$

- What this means is that, if the premises and the negated conclusion are inconsistent, they will remain so after skolemization
- This is all we need for refutation resolution to work

Conversion into clausal form

- **Step 6: Prenex normal form** – moving out all universal quantifiers to the left-hand side of the formula, thereby preserving the original order of the quantifiers

$$\forall x F(x) \vee \forall y G(y) \equiv \forall x \forall y (F(x) \vee G(y))$$
$$\forall x F(x) \wedge \forall y G(y) \equiv \forall x \forall y (F(x) \wedge G(y))$$
$$\forall x F(x) \vee H\{x\} \equiv \forall x (F(x) \vee H\{x\})$$
$$\forall x F(x) \wedge H\{x\} \equiv \forall x (F(x) \wedge H\{x\})$$

- The sequence of quantifiers on the left-hand side is called a **prefix**
- The right-hand side of the formula, which now is quantifier-free, is called a **matrix**

Conversion into clausal form

- **Step 7: Prefix elimination**. Only the matrix remains, for which all variables are implicitly universally quantified
- **Step 8: Conversion to CNF** using distributivity

$$\big(F \lor (G \land H)\big) \equiv \big((F \lor G) \land (F \lor H)\big)$$
$$\big((F \land G) \lor H\big) \equiv \big((F \lor H) \land (G \lor H)\big)$$

- **Step 9: Conversion into a set of clauses** by eliminating the $\land$ operator, which is implicitly assumed between the clauses
- **Step 10: Clause standardization** by renaming variables so that no two clauses contain variables of the same name, using:

$$\forall x(F(x) \land G(x)) \equiv \forall x \forall y(F(x) \land G(y))$$

NB: We are not renaming the same variable within the same clause! Namely, it does not generally hold:

$$\forall x P(x, x) \not\equiv \forall x \forall y P(x, y)$$
$$\forall x\big(P(x) \lor Q(x)\big) \not\equiv \forall x \forall y\big(P(x) \lor Q(y)\big)$$

Conversion into clausal form – example

$$\forall y \forall z \Big(\exists u \big(P(y,u) \lor P(z,u) \big) \to \exists u \forall Q(y,z,u) \Big)$$

- Step 1: Equivalence elimination $\Rightarrow$ OK
- Step 2: Implication elimination

$$\forall y \forall z \big(\neg (\exists u (P(y,u) \lor P(z,u))) \lor \exists u Q(y,z,u) \big)$$

- Step 3: Pushing negations onto atoms

$$\forall y \forall z \big(\forall u (\neg P(y,u) \land \neg P(z,u)) \lor \exists u Q(y,z,u) \big)$$

- Step 4: Variable renaming

$$\forall y \forall z \big(\forall u (\neg P(y,u) \land \neg P(z,u)) \lor \exists v Q(y,z,v) \big)$$

Conversion into clausal form – example

- Step 5: Skolemization

$$\forall y \forall z \big(\forall u (\neg P(y,u) \land \neg P(z,u)) \lor Q(y,z,f(y,z))\big)$$

- Step 6: Prenex normal form

$$\forall y \forall z \forall u \big((\neg P(y,u) \land \neg P(z,u)) \lor Q(y,z,f(y,z))\big)$$

- Step 7: Prefix elimination

$$(\neg P(y,u) \land \neg P(z,u)) \lor Q(y,z,f(y,z))$$

- Step 8: Conversion to CNF

$$\big(\neg P(y,u) \lor Q(y,z,f(y,z))\big) \land \big(\neg P(z,u) \lor Q(y,z,f(y,z))\big)$$

Conversion into clausal form – example

- Step 9: Conversion into a set of clauses

$$\{\neg P(y,u) \lor Q(y,z,f(y,z)), \neg P(z,u) \lor Q(y,z,f(y,z))\}$$

- Step 10: Standardization (with $u \to v$, $y \to w$, $z \to x$)

$$\{\neg P(y,u) \lor Q(y,z,f(y,z)), \neg P(x,v) \lor Q(w,x,f(w,x))\}$$

Unification

- The operation of **converting two expressions into the same form**
- All variables in a clause are universally quantified, so the rule of universal instantiation ($\forall$ elimination) holds: if a variable is substituted by any term, the resulting formula will be the logical consequence of the original formula

Examples

- $S(x) \Rightarrow S(John) \Leftarrow S(John)$, by substitution of variable x with the term $John$, which we denote as $\{John/x\}$
- $S(x) \Rightarrow S(z) \Leftarrow S(y)$, with $\{z/x, z/y\}$
- $Q(x,a) \Rightarrow Q(f(y),a) \Leftarrow Q(f(y),z)$, with $\{f(y)/x, a/z\}$
- $Q(f(x),x) \Rightarrow Q(f(a),a) \Leftarrow Q(y,a)$, with $\{f(a)/y, a/x\}$ resulting from a composition of substitutions, $\{f(x)/y\} \circ \{a/x\} = \{f(a)/y, a/x\}$

- Unification **need not always succeed!**
E.g., expressions $P(a)$ and $P(f(x))$ cannot be unified

Substitution

Substitution

Let x_i be FOL variables and t_i FOL terms. The set of pairs

$$\alpha = \{t_1/x_1, t_2/x_2, \ldots, t_n/x_n\}$$

is a substitution of variables x_i with terms t_i, provided $x_i \neq x_j$ for $i \neq j$ and $t_i \neq x_i$ for $i = 1, \ldots, n$.

- Application of substitution α to an expression K: all occurrences of the variable x_i in K is replaced with the expression t_i
- The resulting expression, denoted $K\alpha$, is an instance of expression K E.g., $K = P(x, f(y))$, $\alpha = \{a/x, b/y\}$: $K\alpha = P(a, f(b))$
- For an empty substitution ε, we have $K\varepsilon = K$
- **NB:** Only variables can be substituted!

Composition of substitutions

- Composition of substitutions α and β, denoted $\alpha \circ \beta$, is a substitution that satisfies $K(\alpha \circ \beta) = (K\alpha)\beta$ for all K

Deriving the composition of substitutions

Given two substitutions:
$\alpha = \{t_1/x_1, t_2/x_2, \ldots, t_n/x_n\}$ and $\beta = \{s_1/y_1, s_2/y_2, \ldots, s_m/y_m\}$.

Construct the following sets:

$$S_1 = \{t_1\beta/x_1, t_2\beta/x_2, \ldots, t_n\beta/x_n, \underbrace{s_1/y_2, s_2, y_2, \ldots, s_m/y_m}_{\beta}\}$$

$$S_2 = \{t_i\beta/x_i \mid t_i\beta/x_i \in S_1, t_i\beta = x_i\}$$
$$S_3 = \{s_i/y_i \mid s_i/y_i \in S_1, y_i \in \{x_1, \ldots, x_n\}\}$$

Composition of substitutions is given by:

$$\alpha \circ \beta = S_1 \setminus S_2 \setminus S_3$$

Composition of substitutions – example

- We are given the substitutions:

$$\alpha = \{z/u, h(u)/w\}$$
$$\beta = \{a/u, z/w, u/z\}$$

- Let's derive the composition $\alpha \circ \beta$:

$$\begin{aligned} S_1 =& \{z\beta/u, h(u)\beta/w, a/u, z/w, u/z\} \\ =& \{u/u, h(a)/w, a/u, z/w, u/z\} \\ S_2 =& \{u/u\} \\ S_3 =& \{a/u, z/w\} \\ \alpha \circ \beta =& S_1 \setminus S_2 \setminus S3 = \{h(a)/w, u/z\} \end{aligned}$$

- Let's show that for $K = P(u, w, f(z))$ we have $K(\alpha \circ \beta) = (K\alpha)\beta$:

$$\begin{aligned} P(u, w, f(z))(\alpha \circ \beta) =& P(u, h(a), f(u)) \\ \big(P(u, w, f(z))\alpha\big)\beta =& P(z, h(u), f(z))\beta = P(u, h(a), f(u)) \end{aligned}$$

Unification

- Expressions K_1 and K_2 can be converted to the same form if a substitution γ exists such that:

$$\boxed{K_1\gamma = K_2\gamma}$$

- Substitution γ is called an unifier, and we say that K_1 and K_2 have been unified by γ
- Expression $K_1\gamma$ or $K_2\gamma$ is called a common instance
- Two expressions can have more than one unifier

Example

Atoms $P(x)$ and $P(y)$ have the following unifiers:

- $\gamma_1 = \{b/x, b/y\}$, which gives the common instance $P(b)$
- $\gamma_2 = \{z/x, z/y\}$, which gives the common instance $P(z)$

Instance $P(z)$ is a more general common instance than $P(b)$. Why?

Most general unifier (MGU)

- We are after unifiers that give **the most general common instance**, because this makes the conclusion more general, and consequently ensures the completeness of the inference method

Most general unifier (MGU)

Substitution δ is the most general unifier (MGU) iff for every unifier γ of K_1 and K_2 there is a substitution θ for which $\gamma = \delta \circ \theta$ holds.

- Intuitively, γ is a less general unifier that can be obtained from the most general unifier δ by additional substitution θ

Example

Unifiers for $P(x)$ and $P(y)$:

- $\delta = \{y/x\}$ (MGU)
- $\gamma = \{b/x, b/y\} = \delta \circ \theta$, where $\theta = \{b/y\}$

MGU algorithm – example

- Find the MGU of the following expressions:

$$K_1 = P(g(u), z, f(z)) \qquad K_2 = P(x, y, f(b))$$

- Step 1:
  - $\{g(u)/x\}$ unifies the first subexpressions of K_1 and K_2 that don't agree
  - $K_1\{g(u)/x\} = P(g(u), z, f(z))$
  - $K_2\{g(u)/x\} = P(g(u), y, f(b))$
- Step 2:
  - $\{y/z\}$ unifies the next subexpressions that don't agree
  - composition $\{g(u)/x\} \circ \{y/z\} = \{g(u)/x, y/z\}$
  - $K_1\{g(u)/x, y/z\} = P(g(u), y, f(y))$
  - $K_2\{g(u)/x, y/z\} = P(g(u), y, f(b))$
- Step 3:
  - $\{b/y\}$ unifies the last subexpressions that don't agree
  - composition $\{g(u)/x, y/z\} \circ \{b/y\} = \{g(u)/x, b/z, b/y\} = \delta$
  - $K_1\delta = P(g(u), b, f(b))$
  - $K_2\delta = P(g(u), b, f(b))$

MGU algorithm

- **Input**: two expressions, K_1 and K_2
- **Output**: the most general unifier, if K_1 and K_2 can be unified, otherwise an error
- We'll consider the recursive algorithm **MGUNIFIER** (Luger, Stubblefield, 1993; Shinghal, 1992)
- The algorithm encodes expressions as nested lists:

$$P(a,b) \Rightarrow [P,a,b]$$
$$P(f(a),g(x,y)) \Rightarrow [P,[f,a],[g,x,y]]$$

- The first element of a list is called the head, the rest is called tail

$$K = [P,[f,a],[g,x,y]]$$
$$\text{head}(K) = P$$
$$\text{tail}(K) = [[f,a,],[g,x,y]]$$

The MGUNIFIER algorithm

```
function mgUnifier(K1, K2)
  if var(K1) or fun(K1) or pred(K1) or K1 = [] or
     var(K2) or fun(K2) or pred(K2) or K2 = [] then
     if K1 = K2 then return ∅
     if K1 = [] or K2 = [] then return fail
     if var(K1) then
        if K1 ∈ K2 then return fail else return {K2/K1}
     if var(K2) then
        if K2 ∈ K1 then return fail else return {K1/K2}
     return fail        -- neither K1 nor K2 is a variable
  else
     α ← mgUnifier(head(K1), head(K2))
     if α = fail then return fail
     β ← mgUnifier(tail(K1)α, tail(K2)α)
     if β = fail then return fail
     return α ∘ β
```

Unification failing

- MGUNIFIER returns an error in cases when unification is not possible
- E.g.:

K_1	K_2	Unification error
$P(a)$	$P(b)$	Constant symbols don't agree
$P(f(x))$	$P(g(b))$	Function symbols don't agree
$P(x)$	$Q(y)$	Predicate symbols don' agree
$P(a)$	$P(x,b)$	Arities don't match
$P(x)$	$P(f(x))$	Substitution of a variable with a term that contains that very variable

Occurs check

- MGUNIFIER performs the so-called **occurs check**: checks whether the term that substitutes for a variable contains that very variable ($K_1 \in K_2$ and $K_2 \in K_1$ in the pseudocode)
- Without this check unification may produce a circular substitution which yields an infinitely nested expression
- E.g., $K_1 = P(x, x)$ and $K_2 = P(f(y), y)$

$$\begin{aligned}
\alpha = \{f(y)/x\} \Rightarrow K_1\alpha &= P(f(y), f(y)) \\
K_2\alpha &= P(f(y), y) \\
\alpha = \{f(y)/y\} \Rightarrow K_1\alpha &= P(f(f(\cdots f(y)\cdots)), f(f(\cdots f(y)\cdots))) \\
K_2\alpha &= P(f(f(\cdots f(y)\cdots)), f(\cdots f(y)\cdots))
\end{aligned}$$

- Not doing the occurs check makes the resolution rule unsound
- E.g., one could prove $\forall x \exists y P(x, y) \vdash \exists y \forall x P(x, y)$, although $\forall x \exists y P(x, y) \nvDash \exists y \forall x P(x, y)$

Unification of literals

- To do resolution, we'll have to be able to unify literals

Unifying literals

Two literals can be **unified** iff

- either both are negative or both are positive
- the atoms themselves can be unified

- E.g., $K_1 = P(x)$ and $K_2 = P(y)$ or $K_1 = \neg P(x)$ and $K_2 = \neg P(y)$

Unifying complementary literals

Two literals can be **complementary unified** iff

- one of them is negative and the other is positive
- the atoms themselves can be unified

- E.g., $K_1 = P(x)$ and $K_2 = \neg P(y)$ or $K_1 = \neg P(x)$ and $K_2 = P(y)$

Outline

Resolution rule for FOL

- Similar to resolution in PL, with addition of unification
- Refutation resolution works in the same way as in PL
- Parent clauses need to be standardized

Resolution rule on FOL clauses

$$\frac{F_1 \lor \cdots \lor F_i \lor \cdots \lor F_n \qquad\qquad G_1 \lor \cdots \lor G_j \lor \cdots \lor G_m}{F_1\delta \lor \cdots \lor F_{i-1}\delta \lor F_{i+1}\delta \lor \cdots \lor F_n\delta \lor G_1\delta \lor \cdots \lor G_{j-1}\delta \lor G_{j+1}\delta \lor \cdots \lor G_m\delta}$$

where F_i and G_j are literals that can be **complementary unified** and δ is the corresponding most general unifier (MGU).

The resolvent is the disjunction of all the remaining literals from the parent clauses, with **substitution** δ being applied to every literal.

Resolving two unit clauses yields the **empty clause** NIL.

Example 1

- Find the resolvent of:

$$
\begin{array}{ll}
(1) & P(g(y), x, f(z)) \vee Q(z, b) \vee R(x) \\
(2) & S(x, y) \vee \neg P(x, y, f(a))
\end{array}
$$

- Clauses are not standardized: let's rename the variables in the first clause using the substitution $\{w/x, u/y\}$
- Standardized clauses:

$$
\begin{array}{ll}
(1) & P(g(u), w, f(z)) \vee Q(z, b) \vee R(w) \\
(2) & S(x, y) \vee \neg P(x, y, f(a))
\end{array}
$$

- Resolving by complementary literals with MGU $\delta = \{g(u)/x, y/w, a/z\}$ gives:

$$
(3) \quad S(g(u), y) \vee Q(a, b) \vee R(y)
$$

Example 2

(1) *Every student attends lectures.*
(2) *John is a student.*
$\vdash$ *John attends the lectures.*

Conversion into clausal form and refutation resolution:

(1) $\neg S(x) \vee L(x)$
(2) $S(John)$
(3) $\neg L(John)$ (negated goal)
(4) $\neg S(John)$ (from 1 and 3 with $\delta = \{John/x\}$)
(5) NIL (from 2 and 4 with $\delta = \varnothing$)

Factorizing FOL clauses

- As in PL, clauses in FOL have to **factorized** for refutation resolution to be **complete**
- An example of how not doing factorization ruins completeness:

(1) $P(u) \vee P(w)$
(2) $\neg P(x) \vee \neg P(y)$
(3) $P(w) \vee \neg P(y)$ (from 1 and 2 with $\delta = \{u/x\}$)

(we obtain a similar result by resolving on other literals)

- A clause can be factorized iff it contains literals that can be unified
- A clause factorized in that way is called a **factor clause**

(1) $P(u) \vee P(w)$
(2) $\neg P(x) \vee \neg P(y)$
(1') $P(w)$ (factor clause of 1 with $\delta = \{w/u\}$)
(2') $\neg P(y)$ (factor clause of 2 with $\delta = \{y/x\}$)
(3) NIL (from 1' and 2' with $\delta = \{w/y\}$)

Factorizing FOL clauses

Factor clause

Let clause

$$F_1 \lor \cdots \lor \cdots \lor F_i \lor \cdots \lor F_j \lor \cdots \lor F_n$$

contain literals F_i and F_j whose most general unifier is δ. The **factor clause** of this clause is the clause

$$F_1\delta \lor \cdots \lor F_i\delta \lor \cdots \lor F_n\delta$$

- E.g., for the clause

  $$P(x, y, f(b)) \lor S(x, y) \lor P(g(u), w, f(z))$$

  the factor clause is

  $$P(g(u), y, f(b)) \lor S(g(u), y) \qquad \text{(with } \delta = \{g(u)/x, y/w, b/z\}\text{)}$$

- **NB:** One clause can have more than one factor clause

Resolution rule in FOL – refined

- To **retain completeness** of refutation resolution, resolution has to be carried out on both factorized and non-factorized parent clauses (all combinations!)

Resolution on FOL clauses

Let F and G be **parent clauses** (clauses that contain literals that can be complementary unified). The **resolvent** of these clauses is any resolvent obtained by:

(1) resolving F and G

(2) resolving F and factor of G

(3) resolving factor of F and G

(4) resolving factor of F and factor of G

- **NB:** If a parent clause has more than one factor clauses, then all combinations have to be considered

Refutation resolution completeness and FOL undecidability

- As in PL, **refutation resolution is sound and complete**
  - Sound: if $F \wedge \neg G$ derives NIL, then F semantically entails G
  - Complete: if F semantically entails G, then $F \wedge \neg G$ derives NIL
- Thus, logical and deductive consequence are really the same thing
- Completeness was proven by J. A. Robinson (1965), but K. Gödel has earlier proven the existence of a complete method for FOL (1929)
- However, unlike PL, FOL **is not decidable**!

Undecidability of validity in FOL

There exists no algorithm that, given FOL formula F as input, will return "yes" if F is valid and "no" if F is not valid for all FOL formulas.

- FOL undecidability was proven by A. Church i A. Turing (1935)

$\Rightarrow$ there is no algorithm for proving all **logical consequences**

  - namely, by **semantic deduction theorem**, $F \vDash G$ iff $\vDash F \rightarrow G$

Semi-decidability of FOL

- More precisely, FOL is **semi-decidable**:
  - there exists algorithms that return "yes" if F is valid, but if F is not valid, the algorithm may never terminate
- Refutation resolution is one such algorithm
- Semi-decidability limits the power of refutation resolution:
  - If F does semantically entail G, then it will derive NIL
  - If F does not semantically entail G, it may never terminate

Example

It is the case

$$\forall x\big(\forall y P(y) \rightarrow P(x)\big) \not\models \forall x P(x)$$

but refutation resolution can't prove this (it doesn't terminate).

Example: Robot and packages

- A robot is delivering packages. The robot knows that all packages in room 27 are smaller than any of the packages in room 28. A and B are packages. Package A is in room 27 or 28, but the robot doesn't know where exactly. Package B is in room 27 and is not smaller than package A.

- Use refutation resolution to show how the robot can conclude that package A is in room 27
- Knowledge representation:

(1) $\forall x \forall y \Big(\big(P(x) \land P(y) \land I(x,27) \land I(y,28)\big) \rightarrow S(x,y) \Big)$

(2) $P(A) \land P(B)$

(3) $I(A,27) \lor I(A,28)$

(4) $I(B,27) \land \neg S(B,A)$

$\vdash I(A,27)$

Example: Robot and packages

- Premises and the negated goal in clausal form:

(1) $\neg P(x) \vee \neg P(y) \vee \neg I(x, 27) \vee \neg I(y, 28) \vee S(x, y)$

(2) $P(A)$

(3) $P(B)$

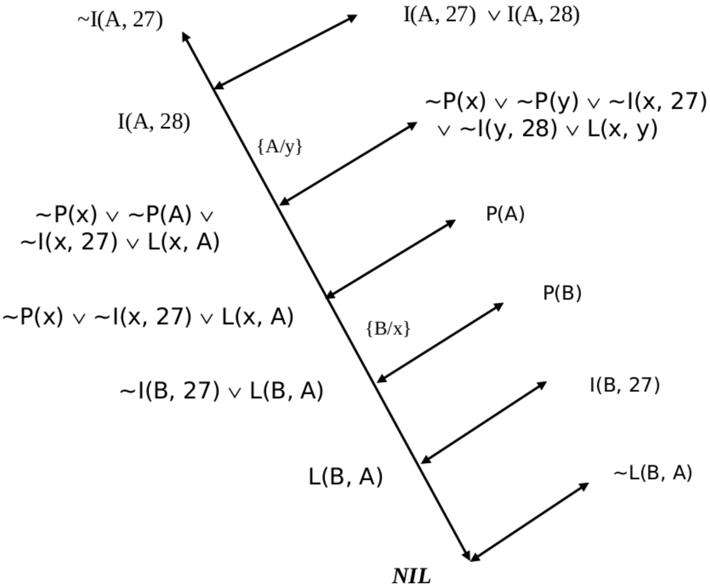
(4) $I(A, 27) \vee I(A, 28)$

(5) $I(B, 27)$

(6) $\neg S(B, A)$

(7) $\neg S(A, 27)$ (negated goal)

Example: Robot and packages



Example: Customs officers and diplomats

Premises

Customs officers searched everyone who entered the country but wasn't a diplomat. Some smugglers who entered the country were searched only by smugglers. No smuggler is a diplomat.

Conclusion

Some customs officers are smugglers.

Example: Customs officers and diplomats

(1) $\forall x\Big(\big(E(x) \land \neg D(x)\big) \rightarrow \exists y\big(O(y) \land S(y,x)\big)\Big)$

(2) $\exists x\Big(M(x) \land E(x) \land \forall y\big(S(y,x) \rightarrow M(y)\big)\Big)$

(3) $\forall x\big(M(x) \rightarrow \neg D(x)\big)$

$\vdash \exists x\big(O(x) \land M(x)\big)$

(1) $\neg E(x) \lor D(x) \lor O(f(x))$

(2) $\neg E(z) \lor D(z) \lor S(f(z), z))$

(3) $M(a)$

(4) $E(a)$

(5) $\neg S(y,a) \lor M(y)$

(6) $\neg M(v) \lor \neg D(v)$

(7) $\neg O(w) \lor \neg M(w)$

Example: Customs officers and diplomats

(8)	$\neg E(x) \lor D(x) \lor \neg M(f(x))$	(from 1 and 7 with $\delta = \{f(x)/w\}$)
(9)	$D(a) \lor \neg M(f(a))$	(from 4 and 8 with $\delta = \{a/x\}$)
(10)	$\neg S(f(a), a) \lor D(a)$	(from 5 and 9 with $\delta = \{f(a)/y\}$)
(11)	$\neg E(a) \lor D(a) \lor D(a)$	(from 2 and 10 with $\delta = \{a/z\}$)
(12)	$D(a)$	(from 4 and 11 with $\delta = \varnothing$)
(13)	$\neg M(a)$	(from 6 and 12 with $\delta = \{a/v\}$)
(14)	NIL	(from 3 and 13 with $\delta = \varnothing$)

Wrap-up

- Proof theory uses **rules of inference** to derive **deductive consequences**, without explicit reference to semantics of logic
- Rules of inference must be **sound** and preferably **complete**
- **Resolution rule** is a simple inference rule that is sound
- Formulas have to be converted to **clausal form** before using resolution
- To resolve FOL literals, we use **unification** to find **substitution** of variables that makes two expressions the same
- A **resolution strategy** simplifies or controls the process of derivation (e.g., **set-of-support strategy**)
- **Refutation resolution** (with standardization, factorization, and a complete strategy) is a sound and complete method for PL and FOL
- **Semi-decidability** of FOL limits the power of resolution

Next topic: Logic programming