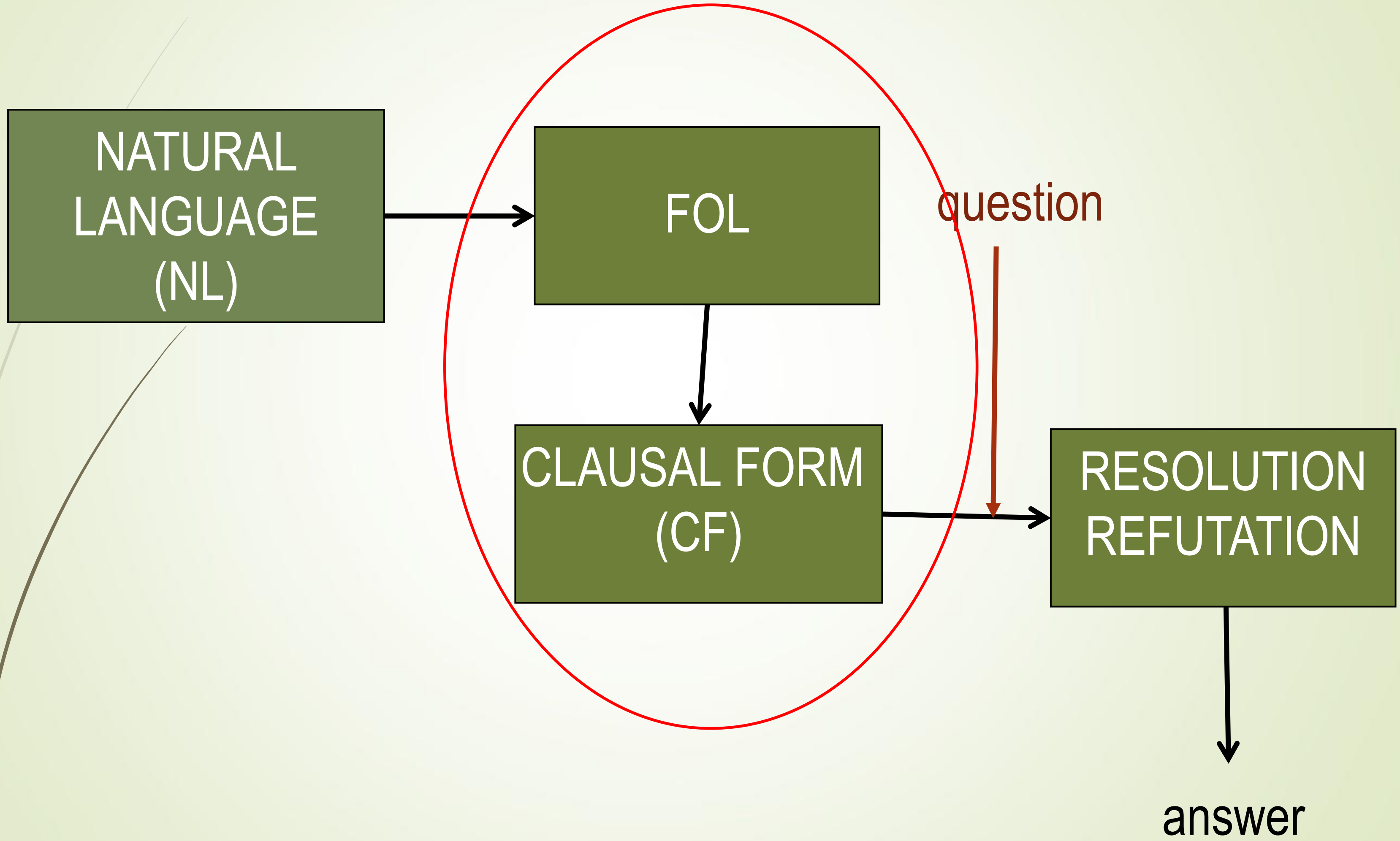
CEID

MSc on DATA DRIVEN COMPUTING AND DECISION MAKING (DDCDM)

AUTOMATED REASONING WITH LOGIC

I. HATZILYGEROUDIS, PROFESSOR EMERITUS

LOGIC AND AUTOMATED REASONING



CLAUSAL FORM OF FOL (1)

BASIC POINTS

- Handling FOL propositions for inference would be complex (due to the many logic symbols) and would cause performance problems.
- The clause form of FOL is a syntactically much simpler form of logic, where all logic symbols have been removed and only the disjunction remains.
- The important thing is that although this form in terms of information and naturalness is subordinate to FOL, in terms of derivability capabilities it is equivalent.
- Also, there is an automatic conversion process

CLAUSAL FORM OF FOL (2)

BASIC DEFINITIONS

- **literal:** an atom (positive literal) or the negation of an atom (negative literal)
- **clause:** a set of literals representing their disjunction

TYPES OF CLAUSES

- empty
- unit
- positive, negative, mixed
- Horn (at most one positive literal)

CLAUSAL FORM OF FOL (3)

CONVERSION TO CLAUSAL FORM (CF)

1. Implication elimination

$(F1 \Rightarrow F2) \rightarrow (\neg F1 \vee F2)$

2. Reduce the scope of negation

$\neg(\neg F) \rightarrow F$

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

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

$\neg(F1 \wedge \ldots \wedge Fn) \rightarrow (\neg F1 \vee \ldots \vee \neg Fn)$

$\neg(F1 \vee \ldots \vee Fn) \rightarrow (\neg F1 \wedge \ldots \wedge \neg Fn)$

CLAUSAL FORM OF FOL (4)

3. Rename variables with the same name that are bound by different quantifiers (usually not applicable)

4. Transform to PNF

5. Eliminate existential quantifiers (Skolemisation)-Replace corresponding variables with
   - Skolem constants or
   - Skolem functions

6. Remove universal quantifiers

7. Transform to CNF

$(F \vee (F1 \wedge \ldots \wedge Fn)) \rightarrow ((F \vee F1) \wedge \ldots \wedge (F \vee Fn))$

CLAUSAL FORM OF FOL (5)

8. Remove logical connectives and write down resulted clauses

9. Rename variables (in case that more than one clause are produced)

CONVERSION TO CF-EXAMPLE (1)

FOL Formula: $(\forall x)\ (a(x) \wedge b(x)) \Rightarrow (\exists\ y)\ d(x, y)$

1. Elimination of implication

   $(\forall x)\ \neg(a(x) \wedge b(x)) \vee (\exists\ y)\ d(x, y)$

2. Reduce the scope of negation

   $(\forall x)\ (\neg a(x) \vee \neg\ b(x)) \vee (\exists\ y)\ d(x, y)$

3. Rename variables (not applicable)
4. Transform to PNF

   $(\forall x)\ (\exists\ y)\ ((\neg a(x) \vee \neg\ b(x)) \vee d(x, y))$

5. Eliminate existential quantifiers

   $(\forall x)\ ((\neg a(x) \vee \neg\ b(x)) \vee d(x, f(x)))$

CONVERSION TO CF-EXAMPLE (2)

6. Remove universal quantifiers

   $((\neg a(x) \vee \neg b(x)) \vee d(x, f(x)))$

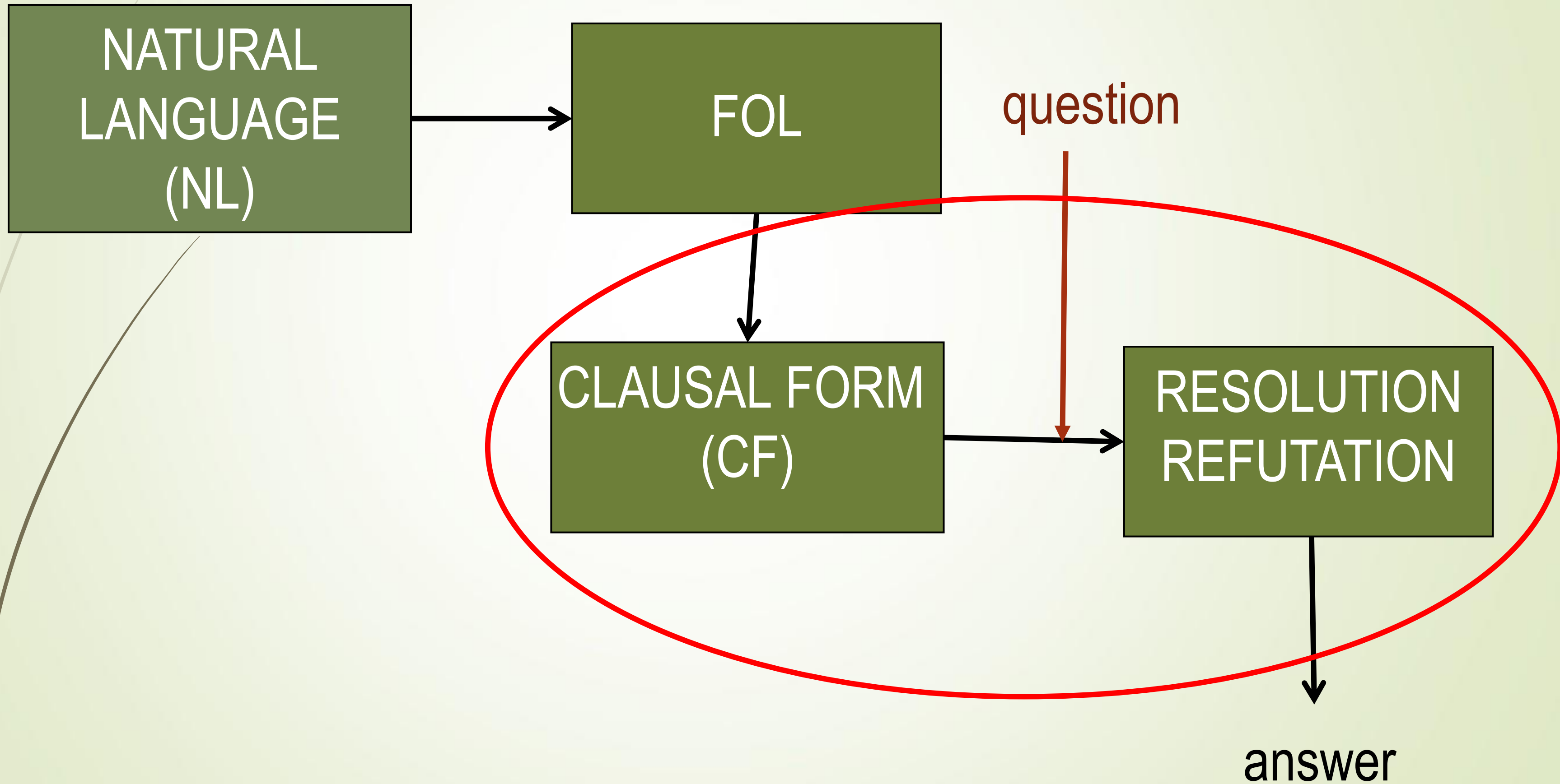
7. Transform to CNF (not applicable)

8. Remove connectives-create clauses

   $\varphi \equiv \{\neg a(x), \neg b(x), d(x, f(x))\}$

9. Rename variables (not applicable)

LOGIC AND AUTOMATED REASONING



SUBSTITUTION (1)

DEFINITION

A substitution θ is a finite set of the form

$\{t_1/v_1, \ldots, t_n/v_n\}$ with $v_i \neq t_i$

where $t_1, \ldots, t_n$ terms → bindings

and $v_1, \ldots, v_n$ variables → bound variables

If none of t_i includes any of v_i then it is called *ground substitution*

<u>Application of a substitution</u> (θ) to an expression (E):

Eθ (*instance* of E)

SUBSTITUTION (2)

Composition of substitutions

$$\theta = \{t_1/x_1, \ldots , t_n/x_n\},\ \sigma = \{u_1/y_1, \ldots , u_m/y_m\}$$

$\theta \circ \sigma$ (ή $\theta\sigma$) $= \{t_1\sigma/x_1, \ldots , t_n\sigma/x_n, u_1/y_1, \ldots , u_m/y_m\}$

except $t_i\sigma/x_i$ with $t_i\sigma = x_i$

and u_i/y_i with $y_i \in \{x_1 , \ldots , x_n \}$

SUBSTITUTION (3)

<u>Example</u>

Let $\theta = \{f(y)/x, y/z\}$, $\sigma = \{a/x, b/y, c/z\}$

Then

1. $\theta \circ \sigma = \{f(b)/x, b/z, \cancel{a/x}, b/y, \cancel{c/z}\}$

2. $\theta \circ \sigma = \{f(b)/x, b/z, b/y\}$

UNIFICATION (1)

- A substitution θ is called a **unifier** of the set {E1, … , En} if E1θ = … = Enθ. The set is called **unifiable**.
- A unifier σ of a set is called **most general unifier** (mgu) if for every other unifier θ of the set there is a substitution λ such that θ = σ ο λ.
- Unification is the process by which we examine whether two expressions can be made <u>syntactically identical</u> by applying some substitution.

UNIFICATION (2)

Terms Unification Rules

1. A constant can only be unified with an identical constant or a variable.

2. A variable is unified with any term unless it is a function containing the variable.

3. A function can only be unified with a function with the same function symbol and unifiable parameters.

Literals Unification Rules

Two literals are unified if they have the same polarity, the same predicate, unifiable terms, and the resulting substitution has no same-variable binding conflicts.

UNIFICATION (3)

Examples

1. p(a, y, z) , p(x, b, z) are unified with mgu σ = {a/x, b/y}
2. q(a, y, z), p(x, b, z) are not unified because $p \neq q$
3. p(a, y, z), ¬p(x, b, z) are not unified due to different polarity
4. p(a, y, z) , p(x, f(a), c) are unified with mgu σ = {a/x, f(a)/y, c/z}

RESOLUTION PRINCIPLE (1)

Resolution Principle

It is an inference rule (IR) that is applied to the clausal form of FOL.

It refers to the production of a "new" clause from two existing ones.

Because this rule alone does not ensure completeness, it is usually accompanied by a simpler rule (or transformation), called factoring.

Factorization acts on one clause and transforms it into another, relying on the unification of literals of the clause.

RESOLUTION PRINCIPLE (2)

Factor: If two or more literals of a clause C have a mgu γ then Cγ is called factor of C (f.C).

Resolution Principle (RP): If L1, L2 are literals of C1, C2 respectively and L1, ¬L2 have a mgu σ then (C1σ - L1σ) ∪ (C2σ - L2σ) is called *binary resolvent* (b.r.) of C1, C2.

Resolvent of two clauses C1, C2 is one of the following:

1. b.r. C1 και C2, **2.** b.r. C1 και f. C2

3. b.r. f.C1 και C2, **4.** b.r. f.C1 και f.C2

RESOLUTION PRINCIPLE (3)

The C1, C2 are called left parent and right parent respectively.

<u>Example</u>

$C1=\{p(x), p(f(y)), r(g(y))\}$, $C2= \{\neg p(f(g(a)), q(b)\}$

$C1$ has factor $C1' = \{p(f(y)), r(g(y))\}$, while C2 does not have a factor

So, because «p(f(y))» and «$\neg$p(f(g(a))» are resolved with mgu

$\sigma = \{g(a)/y\}$ the resolvent of $C1'$ and $C2$ is produced:

$$C12 = \{r(g(g(a)), q(b))$$

THEOREM PROVING (1)

Theorem: If $S \cup \{\varphi\}$ is inconsistent then $S \models \neg\varphi$. Hence if $S \cup \{\neg\varphi\}$ is inconsistent then $S \models \varphi$, where S is a set of logic formulas.

RESOLUTION REFUTATION

The process for the proof of a theorem φ from a set of axioms S (formulas in S in clausal form) is as follows:

1. $S' = S \cup \{\neg\varphi\}$ ($\neg\varphi$ in clausal form)
2. Apply RP, produce resolvent
3. If resolvent = empty clause, stop (success)
4. Update S' (insert resolvent)
5. Go to step 2.

THEOREM PROVING (2)

<u>Example</u>

S = {$C1$, $C2$} με $C1$ = {p(u), p(v)} , $C2$= {¬p(x), ¬p(y)}

$C1$ has factor the $C1$’ = {p(v)}, while C2 the $C2$’ = {¬ p(y)}

The resolvent of $C1$’ και $C2$’ is produced, which is

$C12$ = { }

Hence S is inconsistent.

Notice that without the use of factors the empty clause cannot be produced, so we cannot make the proof.

EXAMPLE (1)

The following FOL formulas are given

(1) works (george, patras)

(2) works (paul, rio)

(3) master (george, pluto)

(4) master (paul, boby)

(5) $(\forall x)\ (\forall y)\ (\text{works}\ (x, y) \Rightarrow \text{lives}\ (x, y))$

(6) $(\forall x)\ (\forall y)\ (\forall z)\ ((\text{master}\ (x, y) \wedge \text{lives}\ (x, z)) \Rightarrow \text{lives}\ (y, z))$

where x, y, z are variables.

(α)

works(george,patra)

works(paul,rio)

master(george,pluto)

master(paul,boby)

$\neg\text{works}(x1,y1) \vee \text{lives}(x1,y1)$

$\neg\text{master}(x2,y2) \vee \neg\text{lives}(x2,z) \vee \text{lives}(y2,z)$

$\neg\text{lives}(x,\text{rio})$

(α) Convert them to CF.

(β) Using Resolution Refutation prove that “lives (pluto, patras)”.

(γ) Using Resolution Refutation , find the values of ‘x’ for which “$(\exists x)$ lives (x, rio)” becomes true.

EXAMPLE (2)

(β)

master(george,pluto) $\quad \neg$master(x2,y2) $\vee$ $\neg$lives(x2,z) $\vee$ lives(y2,z) $\quad \neg$lives(pluto,patra))

σ1 = {pluto/y2, patra/z}

$\neg$master(x2,pluto) $\vee$ $\neg$lives(x2,patra)

σ2 = {george/x2}

$\neg$lives(george,patra) $\quad \neg$works(x1,y1) $\vee$ lives(x1,y1)

σ3 = {george/x1, patra/y1}

works(george,patra) $\quad \neg$works(george,patra)

$\blacksquare$

EXAMPLES with use of equality (1)

Sometimes, it is required to use the equality predicate ('=') in the FOL formulas, used in in infix notation.

Example 1:

- Let a, b, c three constants, and S the following set of FOL:

$$S = \{ a = b, b = c, \neg(a=c) \}$$

- Obviously, S is inconsistent. However, resolution cannot produce the empty clause.

EXAMPLES with use of equality (2)

Example 2:

Let a, b, c three constants, P is a predicate and S the following set of FOL:

$$S = \{ a = b, P(a), \neg P(b) \}$$

Obviously, S is inconsistent. However, resolution cannot produce the empty clause.

EXAMPLES with use of equality (3)

To handle equality in the right way and produce right results, we need to add in our FOL knowledge base the following axioms:

- E1. $\forall x\,(x = x)$
- E2. $\forall x \forall y\,(x=y \Rightarrow y=x)$
- E3. $\forall x \forall y \forall z\,(x=y \land y=z \Rightarrow x=z)$

EXAMPLES with use of equality (4)

Also, for each function *f* with n arguments add the axiom:

- E4. $\forall x_1 \ldots \forall x_n \forall y_1 \ldots \forall y_n (x_1=y_1 \wedge \ldots \wedge x_n=y_n \Rightarrow f(x_1 , \ldots , x_n) = f(y_1 , \ldots , y_n))$

and for each predicate P with n terms add the axiom:

- E5. $\forall x_1 \ldots \forall x_n \forall y_1 \ldots \forall y_n (x_1=y_1 \wedge \ldots \wedge x_n=y_n) \Rightarrow P(x_1 , \ldots , x_n) \equiv P(y_1 , \ldots , y_n))$

After that, equality ('=') can be used in resolution like a regular predicate.

EXAMPLES with use of equality (5)

Prove that S = { father-of(John) = Bill, ∀x (married(father-of(x), mother-of(x)), ¬married(Bill, mother-of(John)) } is unsatisfiable.

We introduce axiom for predicate «married»:

$\forall x1,x2\ \forall y1,y2\ (x1=y1 \wedge x2=y2) \Rightarrow married(x1,x2) \Leftrightarrow married(y1,y2)$

which is analyzed in the following two axioms:

1. $\forall x1,x2\ \forall y1,y2\ (x1=y1 \wedge x2=y2) \Rightarrow (married(x1,x2) \Rightarrow married(y1,y2))$

and

2. $\forall x1,x2\ \forall y1,y2\ (x1=y1 \wedge x2=y2) \Rightarrow (married(y1,y2) \Rightarrow married(x1,x2))$

EXAMPLES with use of equality (6)

1. father-of(John) = Bill
2. $\forall x$ (married(father-of(x), mother-of(x))
3. $\neg$married(Bill, mother-of(John))
4. $\forall x\ (x = x)$
5. $\forall x \forall y\ (x=y \ \Rightarrow y=x)$
6. $\forall x \forall y \forall z\ (x=y \wedge y=z \ \Rightarrow x=z)$
7. $\forall x\ \forall y$ (x=y) => (father-of(x) => father-of(y))
8. $\forall x\ \forall y$ (x=y) => (mother-of(x) => mother-of(y))
9. $\forall$x1,x2 $\forall$y1,y2 (x1=y1 $\wedge$ x2=y2) => (married(x1,x2) => married(y1,y2))
10. $\forall$x1,x2 $\forall$y1,y2 (x1=y1 $\wedge$ x2=y2) => (married(y1,y2) => married(x1,x2))

EXAMPLES with use of equality (7)

Clausal form of (9) is produced as follows (universal quantifiers are removed in advance for the sake of simplicity, given that there are no existential quantfiers):

1. (x1=y1 $\wedge$ x2=y2) => (married(x1,x2) => married(y1,y2))
2. (x1=y1 $\wedge$ x2=y2) => ($\neg$married(x1,x2) $\vee$ married(y1,y2))
3. $\neg$ (x1=y1 $\wedge$ x2=y2) $\vee$ ($\neg$married(x1,x2) $\vee$ married(y1,y2))

and finally

{$\neg$ (x1=y1) , $\neg$(x2=y2), $\neg$married(x1,x2), married(y1,y2)}

Acting similarly for (10), we get:

{$\neg$ (x1=y1), $\neg$(x2=y2), $\neg$married(y1,y2), married(x1,x2)}

EXAMPLES with use of equality (8)

1. {father-of(John) = Bill}
2. { (married(father-of(x), mother-of(x)) }
3. {¬married(Bill, mother-of(John))}
4. {y = y}
5. {¬(z=w), w=z }
6. {¬(r=s), ¬(s=t), r=t }
7. {¬(x1=y1),¬ father-of(x1), father-of(y1)}
8. {¬(x2=y2), ¬ mother-of(x2), mother-of(y2)}
9. {¬(x3=y3), ¬(x4=y4), ¬married(x3,x4), married(y3,y4)}
10. {¬ (x5=y5), ¬(x6=y6), ¬married(y5,y6), married(x5,x6)}

EXAMPLES with use of equality (9)

Εφαρμογή της αντίφασης της επίλυσης:

3.{ $\neg$married(Bill, mother-of(John)) }   9.{ $\neg(x_3=y_3)$, $\neg(x_4=y_4)$, $\neg$married(x_3,x_4), married(y_3,y_4) }

{Bill/y_3, mother-of(John)/y_4}

11.{ $\neg(x_3=$Bill), $\neg(x_4=$mother-of(John)), $\neg$married(x_3,x_4) }   1.{ father-of(John) = Bill }

{father-of(John)/x_4}

12.{ $\neg(x_4=$mother-of(John)), $\neg$married(father-of(John), x_4) }   2.{ married(father-of(x), mother-of(x)) }

{John/x, mother-of(x)/x_4 }

13.{ $\neg$(mother-of(John)=mother-of(John)) }   4.{ y=y }

{ }

PARAMODULATION – A RULE FOR HANDLING EQUALITY (1)

<u>Necessity</u>

- The equality relation is: reflexive, symmetric and transitive
- We need additional K axioms to represent the above properties
- Applying the classical solution to $S \cup K$ is inefficient
- There is a need for a specific rule for handling equality

PARAMODULATION – A RULE FOR HANDLING EQUALITY (2)

Example

- C1: P(a), C2: a = b

  Taking advantage of equality axioms → C3 = P(b).

Definition (Equality substitution)

If a clause C includes term t (C[t]) and if we have the unit clause t = s, then a new clause is produced after substituting s for an occurrence of t (indicated as C[s])

PARAMODULATION – A RULE FOR HANDLING EQUALITY (3)

Ground Paramodulation

From C1: L[t] $\vee$ C1’ and C2: t = s $\vee$ C2’

we derive the paramodulant:

L[s] $\vee$ C1’ $\vee$ C2’

<u>Example</u>

From C1 : P(a) $\vee$ Q(b) and C2 : a = b $\vee$ R(b)

we derive P(b) $\vee$ Q(b) $\vee$ R(b)

PARAMODULATION – A RULE FOR HANDLING EQUALITY (4)

General Paramodulation

Substitution before applying paramodulation

<u>Example</u>

From C1 : $P(x) \vee Q(b)$, C2 : $a = b \vee R(b)$,

$\sigma = \{a/x\}$ and C1’: $C1\sigma = P(a) \vee Q(b)$,

We derive the paramodulant of C1 και C2:

$C3' = P(b) \vee Q(b) \vee R(b)$

PARAMODULATION – A RULE FOR HANDLING EQUALITY (5)

Definition

If C1: $L[t] \vee C1'$, C2: $r = s \vee C2'$ and C1 and C2 have no common variables, and σ the mgu of t και r, we can derive the *binary paramodulant* (b.p.):

$$C12: L\sigma[s\sigma] \vee C1'\sigma \vee C2'\sigma$$

where $L\sigma[s\sigma]$ is derived substituting sσ for an occurrence of tσ in Lσ.

- C12 is called *binary paramodulant* (b.p.) of C1 και C2
- The L and r = s are the literals that were paramodulated
- The process is called paramodulation from C2 to C1.

PARAMODULATION – A RULE FOR HANDLING EQUALITY (6)

Example-1

C1: $P(g(f(x))) \vee Q(x)$ και C2: $f(g(b)) = a \vee R(g(c))$

Application of paramodulation

- t: $f(x)$, L[t]: $P(g(f(x)))$
- r: $f(g(b))$, $r = s$: $f(g(b)) = a$
- $\sigma = \{g(b)/x\}$
- $C12 = P(g(a)) \vee Q(g(b)) \vee R(g(c))$

PARAMODULATION – A RULE FOR HANDLING EQUALITY (7)

Example-2

C1: $P(f(x,a), y) \vee R(y)$ και C2: $f(c,a) = g(b) \vee R(g(b))$

Application of paramodulation (three paramodulants)

- $P(g(b),y) \vee R(y) \vee R(g(b))$
- $P(f(x,a), g(b)) \vee R(f(c,a)) \vee R(g(b))$
- $P(f(x,a), f(c,a)) \vee R(g(b))$

PARAMODULATION – A RULE FOR HANDLING EQUALITY (8)

Paramodulant of two clauses C1, C2 is one of the following:

1. (b.p.) C1 and C2, **2.** (b.p.). C1 and f. C2

3. (b.p.) f.C1 and C2, **4.** (b.p.) f.C1 και f.C2

REASONING WITH TABLEAUX METHOD (1)

- The Tableaux method is a process by which we examine whether (prove that) a set of logical formulas is inconsistent.
- It proceeds step-by-step by breaking down complex logical statements into simpler ones, thus making inconsistency checking simpler.
- The proof process is depicted as a tableaux, i.e. a binary tree, the nodes of which are named with logical formulas.

REASONING WITH TABLEAUX METHOD (2)

- We begin by placing the logical formulas and the negation of the formula to be proved at the root of the tree.
- We apply decomposition rules or expansion rules of (complex) logical formulas into simpler ones, thus creating (new) branches and (new) nodes in the tree.
- Branches containing contradictions/clashes are closed and the corresponding nodes are not developed further.
- If there is no open branch, then it means that the proof is successful.

REASONING WITH TABLEAUX METHOD - PROPOSITIONAL LOGIC (1)

- There are two types of complex logical formulas, conjunctive sentences, called a-sentences, and disjunctive sentences, called b-sentences.
- Accordingly, there are rules-a and rules-b for splitting sentences:

Κανόνες-α			Κανόνες-β		
α	α1	α2	β	β1	β2
$P \wedge Q$	P	Q	$\neg(P \wedge Q)$	$\neg P$	$\neg Q$
$\neg(P \vee Q)$	$\neg P$	$\neg Q$	$P \vee Q$	P	Q
$\neg(P \Rightarrow Q)$	P	$\neg Q$	$P \Rightarrow Q$	$\neg P$	Q
$\neg(P \Leftarrow Q)$	$\neg P$	Q	$P \Leftarrow Q$	P	$\neg Q$

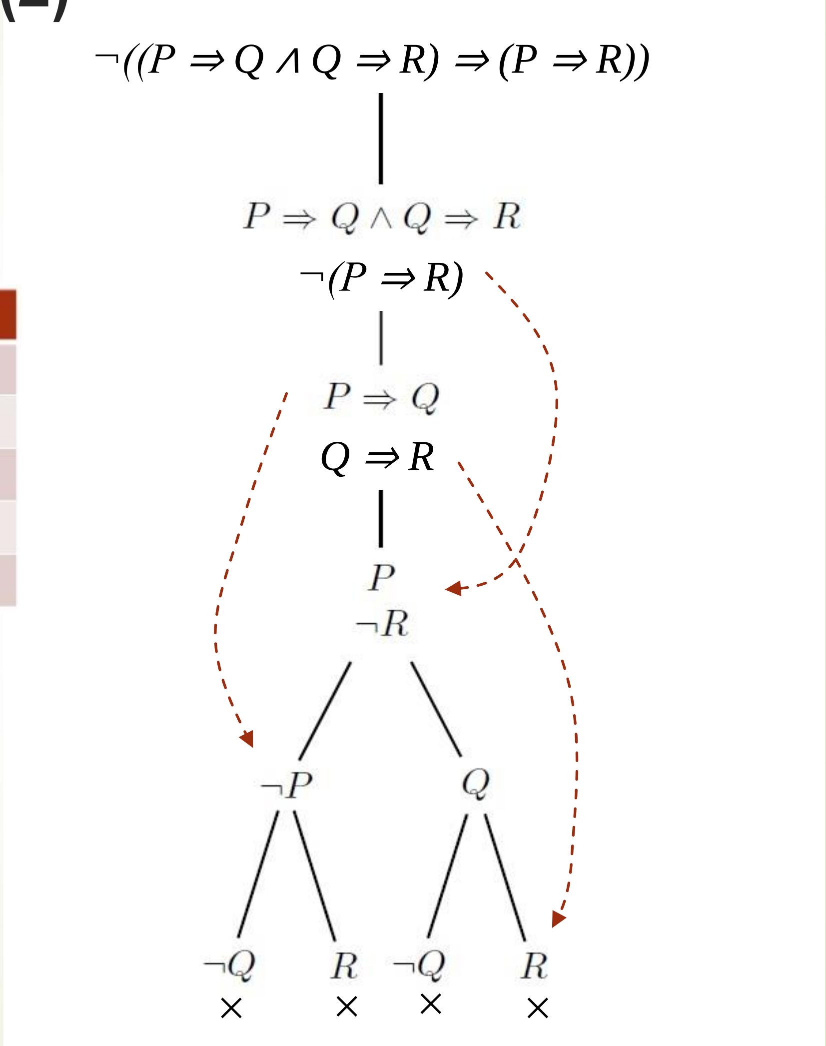
Can be extended for formulas with more than two elements.

REASONING WITH TABLEAUX METHOD - PROPOSITIONAL LOGIC (2)

EXAMPLE-1

Κανόνες-α			Κανόνες-β		
α	α1	α2	β	β1	β2
$P \wedge Q$	P	Q	$\neg(P \wedge Q)$	$\neg P$	$\neg Q$
$\neg(P \vee Q)$	$\neg P$	$\neg Q$	$P \vee Q$	P	Q
$\neg(P \Rightarrow Q)$	P	$\neg Q$	$P \Rightarrow Q$	$\neg P$	Q
$\neg(P \Leftarrow Q)$	$\neg P$	Q	$P \Leftarrow Q$	P	$\neg Q$

Since all leaves have clash (inconsistence), the formula at the root is inconsistent.

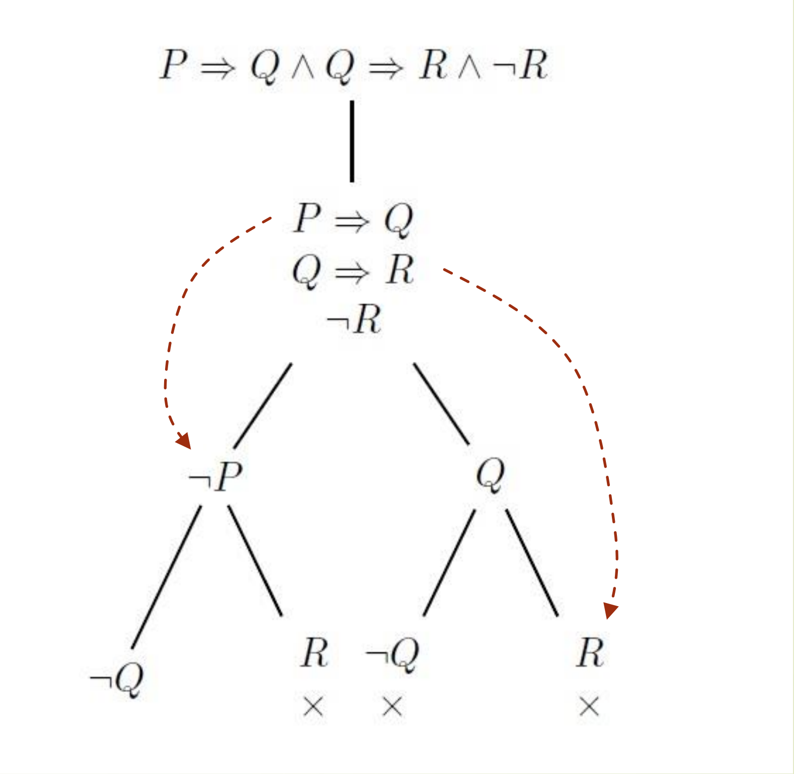


(From Jan van Eijck, Tutorial on Theorem Proving-With corrections)

REASONING WITH TABLEAUX METHOD - PROPOSITIONAL LOGIC (3)

EXAMPLE-2

Κανόνες-α			Κανόνες-β		
α	α1	α2	β	β1	β2
$P \wedge Q$	P	Q	$\neg(P \wedge Q)$	$\neg P$	$\neg Q$
$\neg(P \vee Q)$	$\neg P$	$\neg Q$	$P \vee Q$	P	Q
$\neg(P \Rightarrow Q)$	P	$\neg Q$	$P \Rightarrow Q$	$\neg P$	Q
$\neg(P \Leftarrow Q)$	$\neg P$	Q	$P \Leftarrow Q$	P	$\neg Q$



Since there is a leaf without a clash (inconsistence), the formula at the root is consistent.

(From Jan van Eijck, Tutorial on Theorem Proving)

REASONING WITH TABLEAUX METHOD - PREDICATE LOGIC (1)

- There are two additional types of complex logical sentences, the universal ones, which use a universal quantifier ($\forall$) and are called c-sentences, and the existential ones, which use an existential quantifier ($\exists$) and are called d-sentences.
- Accordingly, there are rules-c and rules-d for splitting sentences :

Κανόνες-γ		Κανόνες-δ	
γ	γ1(t)	δ	δ1(c)
$\forall x F$	$F[t/x]$	$\exists x F$	$F[c/x]$
$\neg(\exists x F)$	$\neg F[t/x]$	$\neg(\forall x F)$	$\neg F[c/x]$

t: any ground term
c: constant not existing in the branch

F[t/x] means replacement of x with t in F.

REASONING WITH TABLEAUX METHOD - PREDICATE LOGIC (2)

ΠΑΡΑΔΕΙΓΜΑ-3

$$\neg[\forall x(p(x) \rightarrow q(x)) \rightarrow (\forall x\, p(x) \rightarrow \forall x\, q(x))]$$

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

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

$$\forall x(p(x) \rightarrow q(x)),\ \forall x\, p(x),\ \neg q(a)$$

$$\forall x(p(x) \rightarrow q(x)),\ p(a),\ \neg q(a)$$

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

$$\neg p(a), p(a), \neg q(a) \qquad q(a), p(a), \neg q(a)$$

$$\times \qquad\qquad \times$$

LOGIC IS MONOTONIC

$$S \models \varphi \Rightarrow (S \cup y) \models \varphi$$

Suppose we have the axioms (knowledge base-KB):

$(\forall x)\ (bird(x) \Rightarrow flies(x))$
bird(Twiti)

Convert them in CF:

$\{\neg\ bird(x) \vee flies(x)\}$ (1)
$\{bird(Twiti)\}$ (2)

We want to prove: flies(Twiti)

We use resolution refutation:

We get the negation : $\neg$flies(Twiti)

Convert it in CF: $\{\neg$flies(Twiti)$\}$

MONOTONICITY-PROOF EXAMPLE (1)

Then KB becomes:

$\{\neg bird(x) \vee fliesx)\}$ (1)
$\{bird(Twiti)\}$ (2)
$\{\neg flies(Twiti)\}$ (3)

Resolve (1) and (2) and we get: $\{flies(Twiti)\}$ (4)
με $\sigma = \{Twiti/x\}$

Resolve (3) and (4) and we get: { } (empty clause)

So, KB became inconsistent by introducing "¬flies(Twiti)", hence "flies(Twiti)" is true: is logically implied from KB.

MONOTONICITY-PROOF EXAMPLE (2)

But we are informed that Twiti is a penguin and that penguins, while they are birds, do not fly. So, we capture the new knowledge with the logical expressions on the right.

$(\forall x)\ (penguin(x) \Rightarrow bird(x))$
$(\forall x)\ (penguin(x) \Rightarrow \neg flies(x))$
$penguin(Twiti)$

We convert them into clausal form and insert them into the KB:

$\{\neg\ bird(x) \vee flies(x)\}$	(1)
$\{bird(Twiti)\}$	(2)
$\{\neg\ penguin(y) \vee bird(y)\}$	(3)
$\{\neg\ penguin(z) \vee \neg\ flies(z)\}$	(4)
$\{penguin(Twiti)\}$	(5)

MONOTONICITY-PROOF EXAMPLE (3)

Let say that we want to prove again that: flies(Twiti)

We find that it is again proved by means of the same clauses.

Let say that we want now to prove that: ¬flies(Twiti)

It is easy to see that this is also proved through the new clauses introduced in KB.

This means that new knowledge that conflicts with older knowledge cannot invalidate it.