

# Computational Logic

## Automated Deduction Fundamentals

1

## Elements of First-Order Predicate Logic

---

First Order Language:

- An *alphabet* consists of the following classes of symbols:
  1. *variables* denoted by  $X, Y, Z, Boo, \dots$ , (infinite)
  2. *constants* denoted by  $1, a, boo, john, \dots$ ,
  3. *functors* denoted by  $f, g, +, -, \dots$ ,
  4. *predicate symbols* denoted by  $p, q, dog, \dots$ ,
  5. *connectives*, which are:  $\neg$  (negation),  $\vee$  (disjunction),  $\wedge$  (conjunction),  $\rightarrow$  (implication) and  $\leftrightarrow$  (equivalence),
  6. *quantifiers*, which are:  $\exists$  (there exists) and  $\forall$  (for all),
  7. *parentheses*, which are: ( and ) and the *comma*, that is: “,”.
- Each functor and predicate symbol has a fixed *arity*, they are often represented in *Functor/Arity* form, e.g.  $f/3$ .
- A constant can be seen as a functor of arity 0.
- Propositions are represented by a predicate symbol of arity 0.

2

## Important: Notation Convention Used

---

(A bit different from standard notational conventions in logic, but good for compatibility with LP systems)

- Variables: start with a capital letter or a “\_” ( $X, Y, \_a, \_1$ )
- Atoms, functors, predicate symbols: start with a lower case letter or are enclosed in ‘ ’ ( $f, g, a, 1, x, y, z, 'X', '_1'$ )

3

## Terms and Atoms

---

We define by induction two classes of strings of symbols over a given alphabet.

- The class of *terms*:
  - ◇ a variable is a term,
  - ◇ a constant is a term,
  - ◇ if  $f$  is an  $n$ -ary functor and  $t_1, \dots, t_n$  are terms then  $f(t_1, \dots, t_n)$  is a term.
- The class of *atoms* (different from LP!):
  - ◇ a proposition is an atom,
  - ◇ if  $p$  is an  $n$ -ary pred. symbol and  $t_1, \dots, t_n$  are terms then  $p(t_1, \dots, t_n)$  is an atom,
  - ◇ true and false are atoms.
- The class of Well Formed Formulas (WFFs):
  - ◇ an atom is a WFF,
  - ◇ if  $F$  and  $G$  are WFFs then so are  $\neg F, (F \vee G), (F \wedge G), (F \rightarrow G)$  and  $(F \leftrightarrow G)$ ,
  - ◇ if  $F$  is a WFF and  $X$  is a variable then  $\exists X F$  and  $\forall X F$  are WFF.
- Literal: positive or negative (non-negated or negated) atom.

4

## Examples

---

### Examples of Terms

- Given:
  - ◇ constants:  $a, b, c, 1, \text{spot}, \text{john}...$
  - ◇ functors:  $f/1, g/3, h/2, +/3...$
  - ◇ variables:  $X, L, Y...$
- Correct:  $\text{spot}, f(\text{john}), f(X), +(1,2,3), +(X, Y, L), f(f(\text{spot})), h(f(h(1,2)), L)$
- Incorrect:  $\text{spot}(X), +(1,2), g, f(f(h))$

### Examples of Literals

- Given the elements above and:
  - ◇ predicate symbols:  $\text{dog}/1, p/2, q/0, r/0, \text{barks}/1...$
- Correct:  $q, r, \text{dog}(\text{spot}), p(X, f(\text{john}))...$
- Incorrect:  $q(X), \text{barks}(f), \text{dog}(\text{barks}(X))$

5

## Examples (Contd.)

---

### Examples of WFFs

- Given the elements above
- Correct:  $q, q \rightarrow r, r \leftarrow q, \text{dog}(X) \leftarrow \text{barks}(X), \text{dog}(X), p(X, Y), \exists X (\text{dog}(X) \wedge \text{barks}(X) \wedge \neg q), \exists Y (\text{dog}(Y) \rightarrow \text{bark}(Y))$
- Incorrect:  $q \vee, \exists p$

6

## More about WFFs

---

- Allow us to represent knowledge and reason about it
  - ◇ Marcus was a man  $man(marcus)$
  - ◇ Marcus was a pompeian  $pompeian(marcus)$
  - ◇ All pompeians were romans  $\forall X pompeian(X) \rightarrow roman(X)$
  - ◇ Caesar was a ruler  $ruler(caesar)$
  - ◇ All romans were loyal to Caesar or they hated him  $\forall X roman(X) \rightarrow loyalto(X,caesar) \vee hate(X,caesar)$
  - ◇ Everyone is loyal to someone  $\forall X \exists Y loyalto(X, Y)$
- We can now reason about this knowledge using standard deductive mechanisms.
- But there is in principle no guarantee that we will prove a given theorem.

7

## Towards Efficient Automated Deduction

---

- *Automated deduction is search.*
- Complexity of search: directly dependent on branching factor at nodes (exponentially!).
- It is vital to cut down the branching factor:
  - ◇ Canonical representation of nodes (allows identifying identical nodes).
  - ◇ As few inference rules as possible.

8

## Towards Efficient Automated Deduction (Contd.)

---

### Clausal Form

- The complete set of logical operators ( $\leftarrow, \wedge, \vee, \neg, \dots$ ) is redundant.
- A minimal (canonical) form would be interesting.
- It would be interesting to separate the quantifiers from the rest of the formula so that they did not need to be considered.
- It would also be nice if the formula were flat (i.e. no parenthesis).
- Conjunctive normal form has these properties [Davis 1960].

### Deduction Mechanism

- A good example:  
Resolution – only two inference rules (*Resolution rule* and *Replacement rule*).

9

## Classical Clausal Form: Conjunctive Normal Form

---

- General formulas are converted to:
  - ◇ Set of *Clauses*.
  - ◇ Clauses are in a logical conjunction.
  - ◇ A clause is a disjunction of the form.  $literal_1 \vee literal_2 \vee \dots \vee literal_n$
  - ◇ The  $literal_i$  are negated or non-negated atoms.
  - ◇ All variables are implicitly universally quantified: i.e. if  $X_1, \dots, X_k$  are the variables that appear in a clause it represents the formula:  
$$\forall X_1, \dots, X_k \quad literal_1 \vee literal_2 \vee \dots \vee literal_n$$
- Any formula can be converted to clausal form automatically by:
  1. Converting to Prenex form.
  2. Converting to conjunctive normal form (conjunction of disjunctions).
  3. Converting to Skolem form (eliminating existential quantifiers).
  4. Eliminating universal quantifiers.
  5. Separating conjunctions into clauses.
- The *unsatisfiability* of a system is preserved.

10

## Substitutions

---

- A substitution is a finite mapping from variables to terms, written as  $\theta = \{X_1/t_1, \dots, X_n/t_n\}$  where
  - ◇ the variables  $X_1, \dots, X_n$  are different,
  - ◇ for  $i = 1, \dots, n$   $X_i \neg \equiv t_i$ .
- A pair  $X_i/t_i$  is called a **binding**.
- $\text{domain}(\theta) = \{X_1, \dots, X_n\}$  and  $\text{range}(\theta) = \text{vars}(\{t_1, \dots, t_n\})$ .
- If  $\text{range}(\theta) = \emptyset$  then  $\theta$  is called **ground**.
- If  $\theta$  is a bijective mapping from variables to variables then  $\theta$  is called a **renaming**.
- Examples:
  - ◇  $\theta_1 = \{X/f(A), Y/X, Z/h(b, Y), W/a\}$
  - ◇  $\theta_2 = \{X/a, Y/a, Z/h(b, c), W/f(d)\}$  (ground)
  - ◇  $\theta_3 = \{X/A, Y/B, Z/C, W/D\}$  (renaming)

11

## Substitutions (Contd.)

---

- Substitutions operate on *expressions*, i.e. a term, a sequence of literals or a clause, denoted by  $E$ .
- The application of  $\theta$  to  $E$  (denoted  $E\theta$ ) is obtained by *simultaneously* replacing each occurrence in  $E$  of  $X_i$  by  $t_i$ ,  $X_i/t_i \in \theta$ .
- The resulting expression  $E\theta$  is called an *instance* of  $E$ .
- If  $\theta$  is a renaming then  $E\theta$  is called a *variant* of  $E$ .
- Example:  
 $\theta_1 = \{X/f(A), Y/X, Z/h(b, Y), W/a\}$   
 $p(X, Y, X) \theta_1 = p(f(A), X, f(A))$

12

## Composition of Substitutions

---

- Given  $\theta = \{X_1/t_1, \dots, X_n/t_n\}$  and  $\eta = \{Y_1/s_1, \dots, Y_m/s_m\}$  their *composition*  $\theta\eta$  is defined by removing from the set

$$\{X_1/t_1\eta, \dots, X_n/t_n\eta, Y_1/s_1, \dots, Y_m/s_m\}$$

those pairs  $X_i/t_i\eta$  for which  $X_i \equiv t_i\eta$ , as well as those pairs  $Y_i/s_i$  for which  $Y_i \in \{X_1, \dots, X_n\}$ .

- Example: if  $\theta = \{X/3, Y/f(X, 1)\}$  and  $\eta = \{X/4\}$  then  $\theta\eta = \{X/3, Y/f(4, 1)\}$ .
- For all substitutions  $\theta, \eta$  and  $\gamma$  and an expression  $E$ 
  - $(E\theta)\eta \equiv E(\theta\eta)$
  - $(\theta\eta)\gamma = \theta(\eta\gamma)$ .
- $\theta$  is more general than  $\eta$  if for some  $\gamma$  we have  $\eta = \theta\gamma$ .
- Example:  $\theta = \{X/f(Y)\}$  more general than  $\eta = \{X/f(h(G))\}$

## Unifiers

---

- If  $A\theta \equiv B\theta$ , then
  - $\diamond$   $\theta$  is called a *unifier* of  $A$  and  $B$
  - $\diamond$   $A$  and  $B$  are *unifiable*
- A unifier  $\theta$  of  $A$  and  $B$  is called a *most general unifier (mgu)* if it is *more general* than any other unifier of  $A$  and  $B$ .
- If two atoms are unifiable then they have a most general unifier.
- $\theta$  is *idempotent* if  $\theta\theta = \theta$ .
- A unifier  $\theta$  of  $A$  and  $B$  is *relevant* if all variables appearing either in  $\text{domain}(\theta)$  or in  $\text{range}(\theta)$ , also appear in  $A$  or  $B$ .
- If two atoms are unifiable then they have an mgu which is idempotent and relevant.
- An mgu is unique up to renaming.

## Unification Algorithm

- Non-deterministically choose from the set of equations an equation of a form below and perform the associated action.
  1.  $f(s_1, \dots, s_n) = f(t_1, \dots, t_n) \rightarrow$  replace by  $s_1 = t_1, \dots, s_n = t_n$
  2.  $f(s_1, \dots, s_n) = g(t_1, \dots, t_m)$  where  $f \neq g \rightarrow$  halt with failure
  3.  $X = X \rightarrow$  delete the equation
  4.  $t = X$  where  $t$  is not a variable  $\rightarrow$  replace by the equation  $X = t$
  5.  $X = t$  where  $X \neq t$  and  $X$  has another occurrence in the set of equations  $\rightarrow$ 
    - 5.1 if  $X$  appears in  $t$  then halt with failure
    - 5.2 otherwise apply  $\{X/t\}$  to every other equation
- Consider the set of equations  $\{f(x) = f(f(z)), g(a, y) = g(a, x)\}$ :
  - ◇ (1) produces  $\{x = f(z), g(a, y) = g(a, x)\}$
  - ◇ then (1) yields  $\{x = f(z), a = a, y = x\}$
  - ◇ (3) produces  $\{x = f(z), y = x\}$
  - ◇ now only (5) can be applied, giving  $\{x = f(z), y = f(z)\}$
  - ◇ No step can be applied, the algorithm successfully terminates.

15

## Unification Algorithm revisited

- Let  $A$  and  $B$  be two formulas:
  1.  $\theta = \epsilon$
  2. while  $A\theta \neq B\theta$ :
    - 2.1 find leftmost symbol in  $A\theta$  s.t. the corresponding symbol in  $B\theta$  is different
    - 2.2 let  $t_A$  and  $t_B$  be the terms in  $A\theta$  and  $B\theta$  starting with those symbols
      - (a) if neither  $t_A$  nor  $t_B$  are variables or one is a variable occurring in the other  $\rightarrow$  halt with failure
      - (b) otherwise, let  $t_A$  be a variable  $\rightarrow$  the new  $\theta$  is the result of  $\theta\{t_A/t_B\}$
  3. end with  $\theta$  being an m.g.u. of  $A$  and  $B$

16

## Unification Algorithm revisited (Contd.)

---

- Example:  $A = p(X, X)$   $B = p(f(A), f(B))$

$\theta$	$A\theta$	$B\theta$	Element
$\epsilon$	$p(X, X)$	$p(f(A), f(B))$	$\{X/f(A)\}$
$\{X/f(A)\}$	$p(f(A), f(A))$	$p(f(A), f(B))$	$\{A/B\}$
$\{X/f(B), A/B\}$	$p(f(B), f(B))$	$p(f(B), f(B))$	

- Example:  $A = p(X, f(Y))$   $B = p(Z, X)$

$\theta$	$A\theta$	$B\theta$	Element
$\epsilon$	$p(X, f(Y))$	$p(Z, X)$	$\{X/Z\}$
$\{X/Z\}$	$p(Z, f(Y))$	$p(Z, Z)$	$\{Z/f(Y)\}$
$\{X/f(Y), Z/f(Y)\}$	$p(f(Y), f(Y))$	$p(f(Y), f(Y))$	

17

## Resolution with Variables

---

- It is a *formal system* with:
  - ◇ A first order language with the following formulas:
    - \* Clauses: without repetition, and without an order among their literals.
    - \* The empty clause  $\square$ .
  - ◇ An empty set of axioms.
  - ◇ Two inference rules: *resolution* and *replacement*.

18

## Resolution with Variables (Contd.)

- Resolution:

$$\frac{r_1: A \vee F_1 \vee \dots \vee F_n \quad r_2: \neg B \vee G_1 \vee \dots \vee G_m}{((F_1 \vee \dots \vee F_n)\sigma \vee G_1 \vee \dots \vee G_m)\theta}$$

where

- ◇  $A$  and  $B$  are unifiable with substitution  $\theta$
- ◇  $\sigma$  is a renaming s.t.  $(A \vee F_1 \vee \dots \vee F_n)\sigma$  and  $\neg B \vee G_1 \vee \dots \vee G_m$  have no variables in common
- ◇  $\theta$  is the m.g.u. of  $A\sigma$  and  $B$

The resulting clause is called the *resolvent* of  $r_1$  and  $r_2$ .

- Replacement:  $A \vee B \vee F_1 \vee \dots \vee F_n \Rightarrow (A \vee F_1 \vee \dots \vee F_n)\theta$  where
  - ◇  $A$  and  $B$  are unifiable atoms
  - ◇  $\theta$  is the m.g.u. of  $A$  and  $B$

19

## Basic Properties

- Resolution is *correct* – i.e. all conclusions obtained using it are valid.
- There is no guarantee of directly deriving a given theorem.
- However, resolution (under certain assumptions) is refutation complete: if we have a set of clauses  $K = [C_0, C_1, \dots, C_n]$  and it is inconsistent then resolution will arrive at the empty clause  $\square$  in a finite number of steps.
- Therefore, a valid theorem (or a question that has an answer) is guaranteed to be provable by refutation. To prove “p” given  $K_0 = [C_0, C_1, \dots, C_n]$ :
  1. Negate it ( $\neg p$ ).
  2. Construct  $K = [\neg p, C_0, C_1, \dots, C_n]$ .
  3. Apply resolution steps repeatedly to  $K$ .
- Furthermore, we can obtain answers by composing the substitutions along a path that leads to  $\square$  (very important for realizing Greene’s dream!).
- It is important to use a good method in applying the resolution steps – i.e. in building the resolution tree (or proof tree).
- Again, the main issue is to reduce the branching factor.

20

## Proof Tree

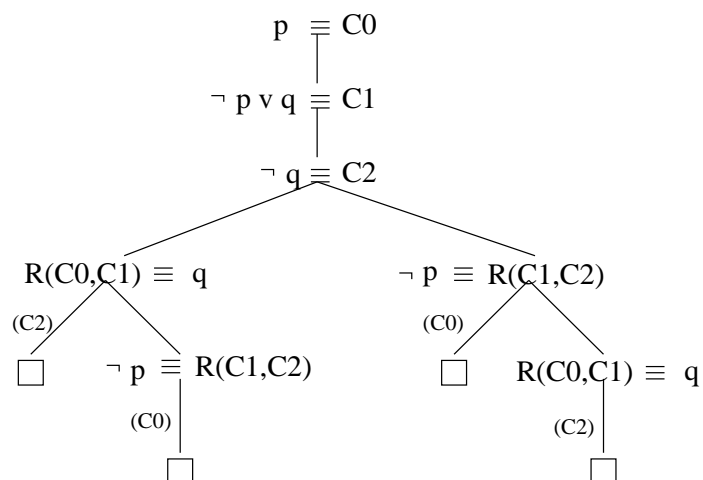
- Given a set of clauses  $K = \{C_0, C_1, \dots, C_n\}$  the proof tree of  $K$  is a tree s.t. :
  - the root is  $C_0$
  - the branch from the root starts with the nodes labeled with  $C_0, C_1, \dots, C_n$
  - the descendent nodes of  $C_n$  are labeled by clauses obtained from the parent clauses using resolution
  - a derivation in  $K$  is a branch of the proof tree of  $K$
- The derivation  $C_0 C_1 \dots C_n F_0 \dots F_m$  is denoted as  $K, F_0 \dots F_m$

21

## Proof Tree (Contd.)

- Example: part of the proof tree for  $K$ , with:

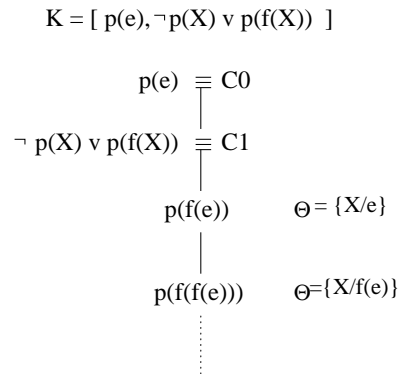
$$K = [ p, \neg p \vee q, \neg q ]$$



22

## Characteristics of the Proof Tree

- It can be infinite:

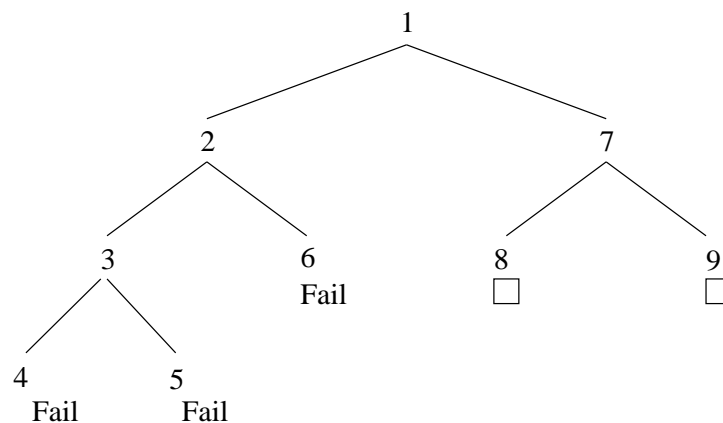


- Even if it is finite, it can be too large to be explored efficiently
- Aim: determine some criteria to limit the number of derivations and the way in which the tree is explored  $\Rightarrow$  strategy
- Any strategy based on this tree is correct: if  $\square$  appears in a subtree of the proof tree of  $K$ , then  $\square$  can be derived from  $K$  and therefore  $K$  is unsatisfiable

23

## General Strategies

- Depth-first with backtracking: First descendant to the left; if failure or  $\square$  then backtrack

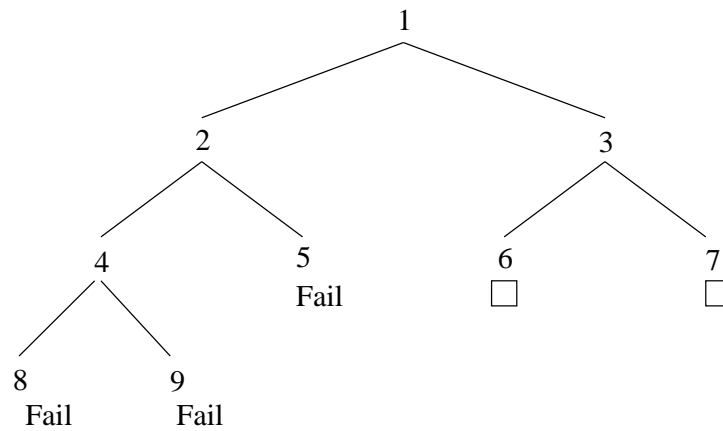


24

## General Strategies (Contd.)

---

- **Breadth first:** all sons of all sibling nodes from left to right



25

## General Strategies (Contd.) (Contd.)

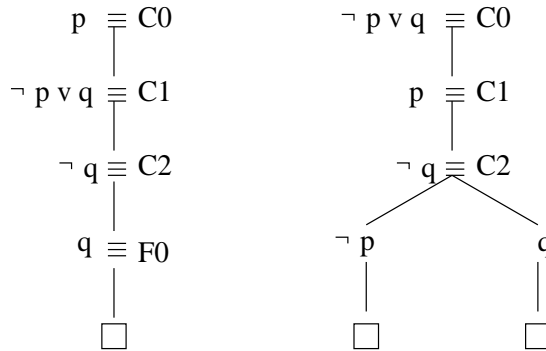
---

- **Iterative deepening**
  - ◇ Advance depth-first for a time.
  - ◇ After a certain depth, switch to another branch as in breadth-first.
  
- **Completeness issues / possible types of branches:**
  - ◇ Success (always finite)
  - ◇ Finite failure
  - ◇ Infinite failure (provably infinite branches)
  - ◇ Non-provably infinite branches

26

## Linear Strategies

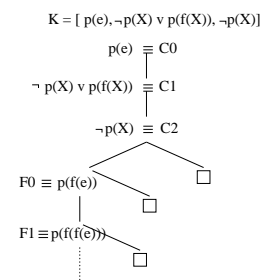
- Those which only explore linear derivations
- A derivation  $K, F_0 \dots F_m$  is linear if
  - ◊  $F_0$  is obtained by resolution or replacement using  $C_0$
  - ◊  $F_i, i < 0$  is obtained by resolution or replacement using  $F_{i-1}$
- Examples:



27

## Characteristics of these Strategies

- 1 If  $\square$  can be derived from  $K$  by using resolution with variables, it can also be derived by linear resolution
  - 2 Let  $K$  be  $K' \cup \{C_0\}$  where  $K'$  is a satisfiable set of clauses, i.e.  $\square$  cannot be derived from  $K'$  by using resolution with variables. If  $\square$  can be derived from  $K$  by using resolution with variables it can also be derived by linear resolution with root  $C_0$ .
- From (1), if the strategy is breadth first, it is complete.
  - From (2), if we want to prove that  $B$  is derived from  $K'$  then we can apply linear resolution to  $K = K' \cup \{\neg B\}$ .
  - Depth first with backtracking is not complete:



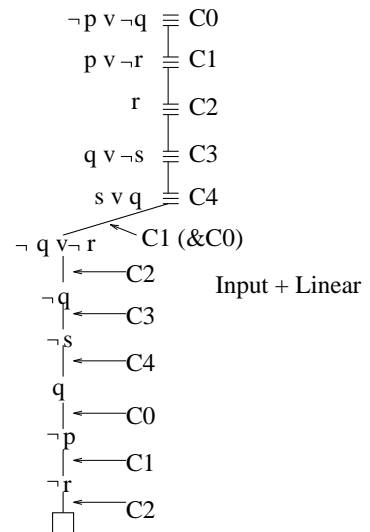
28

## Input Strategies

- Example:

$$K = [\neg p \vee \neg q, p \vee \neg r, r, q \vee \neg s, s \vee q]$$

- Those which only explore input derivations
- A derivation  $K, F_0 \dots F_m$  is input if
  - ◇  $F_0$  is obtained by resolution or replacement using  $C_0$
  - ◇  $F_i, i < 0$  is obtained by resolution or replacement using at least a clause in  $K$



29

## Input Strategies

- In an input derivation, if  $F_{i-1}$  does not appear in any derivation of a successor clause, it can be eliminated from the derivation without changing the result
- If  $F_{i-1}$  appears in the derivation of  $F_j, j > 1, F_{i-1}$  can be allocated in position  $j - 1$
- As a result, we can limit ourselves to linear input derivations without losing any input derivable clause
- Let  $K$  be  $K' \cup \{C_0\}$  where  $\square$  is derived by using resolution with variables,  $C_0$  is a negative Horn clause and all clauses in  $K'$  are positive Horn clauses. There is an input derivation with root  $C_0$  finishing in  $\square$  and in which the replacement rule is not used (Hernschen 1974)
- A *Horn clause* is a clause in which at most one literal is positive:
  - ◇ it is *positive* if precisely one literal is positive
  - ◇ it is *negative* if all literals are negatives
- As a result, in those conditions, a breadth first input strategy is complete, and a depth first input strategy with backtracking is complete if the tree is finite.

30

## Ordered Strategies

---

- We consider a new formal system in which:
  1. clauses are *ordered sets*
  2. ordered resolution of two clauses  
 $A = p_1 \vee \dots \vee p_n$  and  $B = q_1 \vee \dots \vee q_m$   
where  $p_1$  is a positive literal and  $q_1$  is a negative literal is possible iff  $\neg p_1$  and  $\sigma(q_1)$  are unifiable ( $\sigma$  is a renaming, s.t.  $p_1$  and  $\sigma(q_1)$  have no variables in common)
  3. the resolvent of  $A$  and  $B$  is  $\theta(p_2 \vee \dots \vee p_n \vee \sigma(q_2 \vee \dots \vee q_m))$  where  $\theta$  is an m.g.u of  $\neg p_1$  and  $\sigma(q_1)$
- Let  $K = K' \cup \{C_0\}$  be a set of clauses s.t.  $\square$  is derived by using resolution with variables,  $C_0$  is a negative Horn clause and all clauses in  $K'$  are positive Horn clauses with the positive literal in the first place. There is a sorted input derivation with root  $C_0$  arriving at  $\square$ .
- In this context a sorted linear input with:
  - ◇ breadth first: is complete
  - ◇ depth first with backtracking: is complete if the tree is finite