

Resolution and Refutation

York University

Department of Computer Science and Engineering

Overview

- Propositional Logic
 - Resolution
 - Refutation
- Predicate Logic
 - Substitution
 - Unification
 - Resolution
 - Refutation
 - Search space

[ref.: Nilsson- Chap.3]

[Prof. Zbigniew Stachniak's class notes]

Theorems from Logic

[from Mathematical Logic, George Tourlakis]

- Modus Ponens $A, A \rightarrow B \vdash B$
- Cut Rule $A \vee B, \neg A \vee C \vdash B \vee C$
 $A, \neg A \vdash \perp$
- Transitivity of \rightarrow $A \rightarrow B, B \rightarrow C \vdash A \rightarrow C$
- Proof by Contradiction $\Gamma \vdash A \text{ iff } \Gamma + \neg A \vdash \perp$

Resolution in Logic

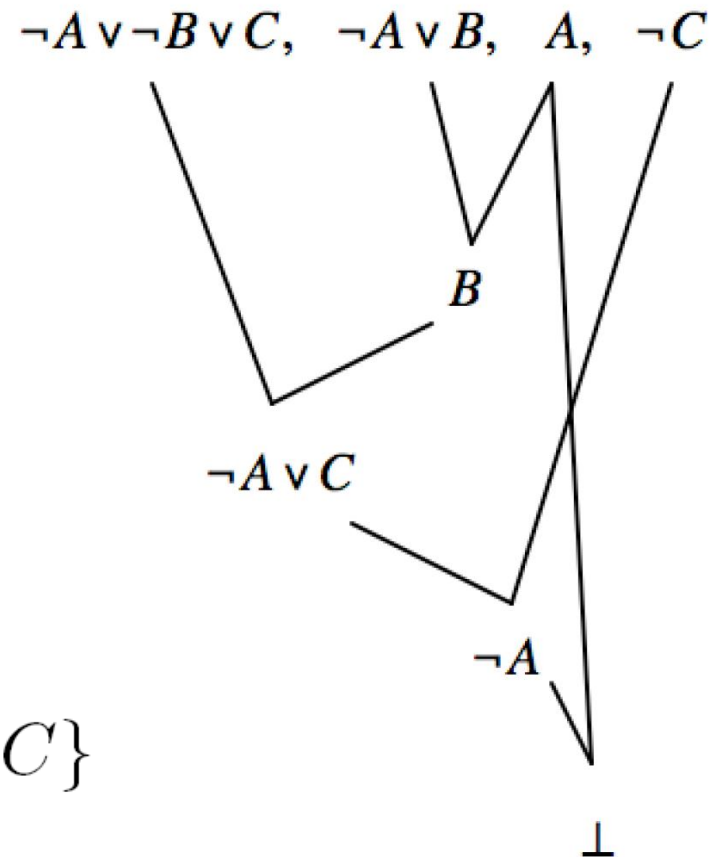
- By A. Robinson (1965)

- Example: Prove

$$A \rightarrow (B \rightarrow C), A \rightarrow B, A \vdash C$$

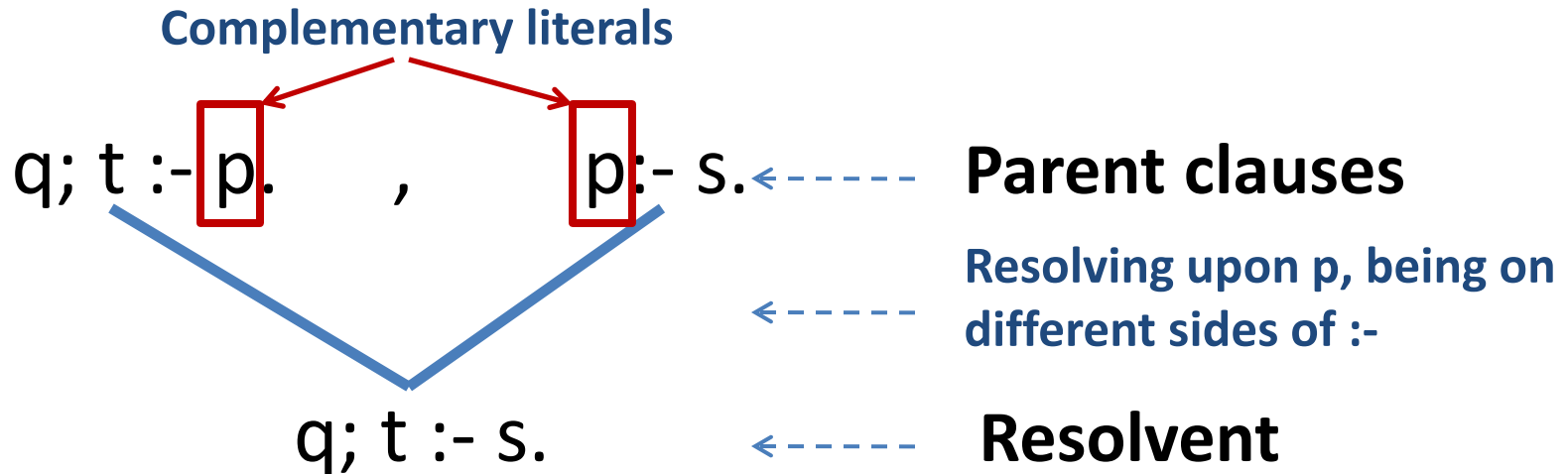
- We need to show that the following set is inconsistent:

$$\{\neg A \vee \neg B \vee C, \neg A \vee B, A, \neg C\}$$



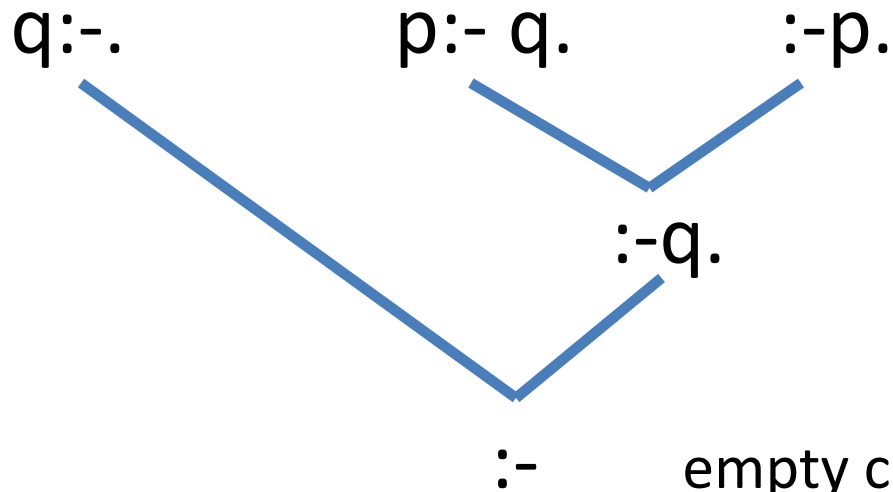
Resolution in Logic Programming

- Program P (facts and rules in clause form)
- Goal G negated and added to program P
- To prove G, we need to show $P + \{\neg G\}$ is inconsistent



Example (1)

- Program $P = \{q:-. , p:-q.\}$
- Query $:-p.$
 - This is already the negated form of our goal!



Refutation

- When resolution is used to prove inconsistency, it is also called refutation. (refute=disprove)
- The above binary tree, showing resolution and resulting in the empty clause, is called a refutation tree.
- NOTE: To avoid potential mistakes, DO NOT RESOLVE UPON MORE THAN ONE LITERAL SIMULTANEOUSLY.

Example (2)

- A1. If Henry has two days off, then if the weather is bad, Henry is not fishing.
- A2. if Henry is not fishing and is not drinking in a pub with his friends, then he is watching TV at home.
- A3. If Henry is working, then he is neither drinking in a pub with his friends nor watching TV at home.
- Q. If Henry is not watching TV at home and he has two days off, then he is drinking in a pub with his friends provided that the weather is bad.

Example (2) (cont.)

- From logical point of view, we want to prove Q , given $A1, A2, A3$. $\{A1, A2, A3\} \vdash Q$.
- By refutation principle, the consistency of $C = \{A1, A2, A3\} \cup \{\neg Q\}$ is examined.
 - Step 1: Represent as propositional formulas
 - Step 2: Represent as clauses
 - Step 3: Determine the consistency of C
 - If C is consistent, answer NO (false)
 - If C is inconsistent, answer YES (true)

Example (2) (cont.)

A1. If Henry has two days off, then if the weather is bad, Henry is not fishing.

A2. if Henry is not fishing and is not drinking in a pub with his friends, then he is watching TV at home.

A3. If Henry is working, then he is neither drinking in a pub with his friends nor watching TV at home.

Q. If Henry is not watching TV at home and he has two days off, then he is drinking in a pub with his friends provided that the weather is bad.

p: H has two days off

q: weather is bad

r: H is fishing

s: H is drinking in a pub with his friends

t: H is watching TV at home

u: H is working

A1. $p \rightarrow (q \rightarrow \sim r)$

A2. $(\sim r \ \& \ \sim s) \rightarrow t$

A3. $u \rightarrow (\sim s \ \& \ \sim t)$

Q. $(\sim t \ \& \ p) \rightarrow (q \rightarrow s)$

Example (2) (cont.)

- Conversion to clause form

$$A1 : p \rightarrow (q \rightarrow \neg r) \Rightarrow \neg p \vee \neg q \vee \neg r \Rightarrow C_1 \text{ is } : \neg p, q, r.$$

$$A2 : (\neg r \wedge \neg s) \rightarrow t \Rightarrow \neg(\neg r \wedge \neg s) \vee t \Rightarrow r \vee s \vee t \Rightarrow C_2 \text{ is } r; s; t : -.$$

$$A3 : u \rightarrow (\neg s \wedge \neg t) \Rightarrow \neg u \vee (\neg s \wedge \neg t) \Rightarrow (\neg u \vee \neg s) \wedge (\neg u \vee \neg t)$$

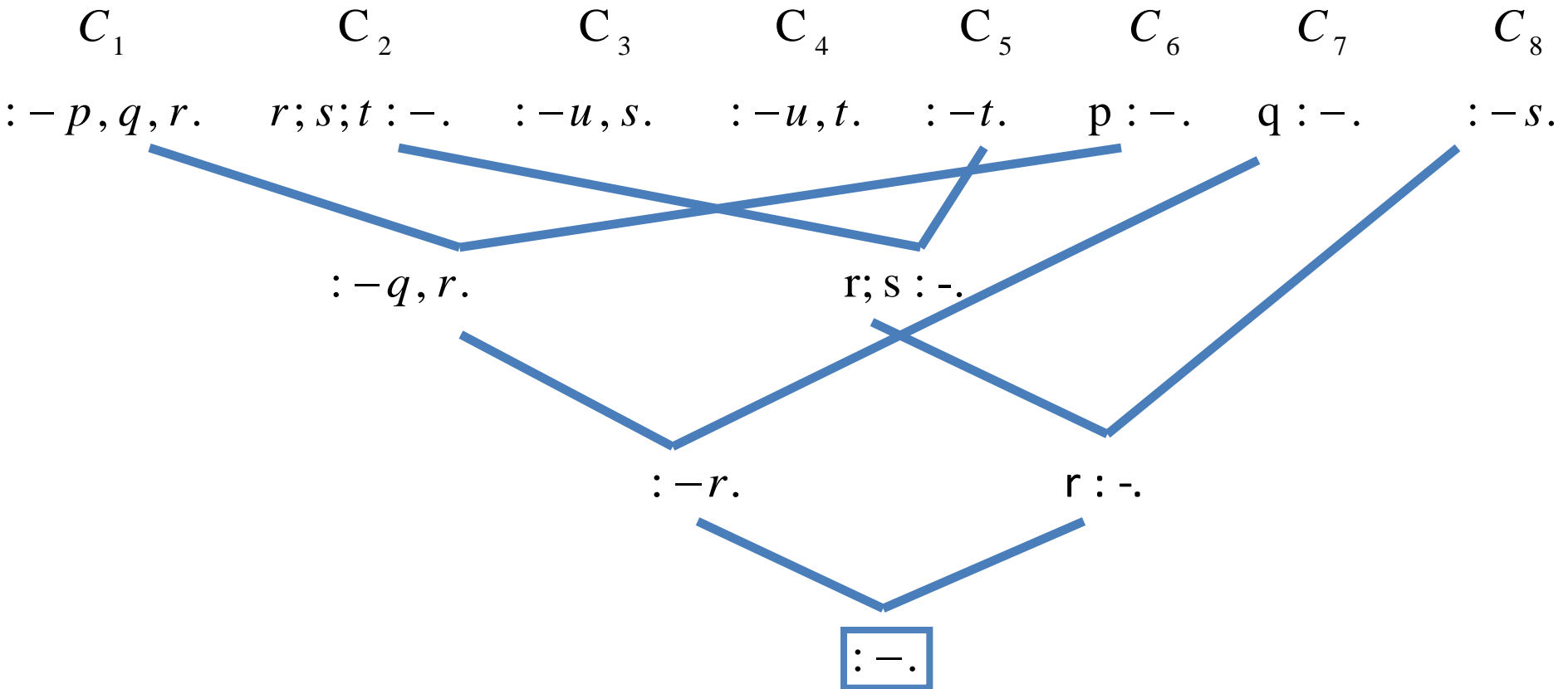
$$\Rightarrow \begin{cases} C_3 \text{ is } : \neg u, s. \\ C_4 \text{ is } : \neg u, t. \end{cases}$$

$$\neg Q : \neg((\neg t \wedge p) \rightarrow (q \rightarrow s)) \Rightarrow (\neg t \wedge p) \wedge \neg(\neg q \vee s) \Rightarrow \neg t \wedge p \wedge q \wedge \neg s$$

$$\Rightarrow \begin{cases} C_5 \text{ is } : \neg t. \\ C_6 \text{ is } p : -. \\ C_7 \text{ is } q : -. \\ C_8 \text{ is } : \neg s. \end{cases}$$

Example (2) (cont.)

- Determining the consistency of $\{C_1, C_2, \dots, C_8\}$



Example (2) (cont.)

- $C = \{C1, C2, \dots, C8\}$ is inconsistent (by resolution/refutation)
- Therefore Q is provable (deducible)
- Answer: YES (true)

- This is how Prolog answers Queries. If the empty string is deduced, Prolog answers YES (or TRUE).

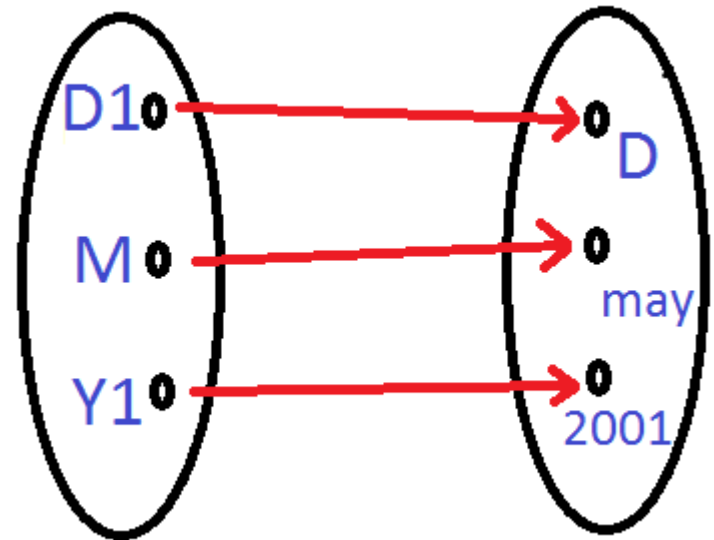
Resolution in Predicate Logic

- A literal in Predicate Logic (PL) is either
 - A positive literal in the form of $p(t_1, \dots, t_k)$ where p is a predicate and t_i are terms
 - Or a negative literal in the form of $\neg p(t_1, \dots, t_k)$
- Two clauses in PL can be resolved upon two complementary unifiable literals
- Two literals are unifiable if a substitution can make them identical.
- Example:
 - `study_hard(X)` and `study_hard(john)`
 - `date(D, M, 2001)` and `date(D1, may, Y1)`

Substitution

- Substitution: is a finite set of pairs of terms denoted as $[X_1/t_1, \dots, X_n/t_n]$ where each t_i is a term and each X_i is a variable.

- Every variable is mapped to a term; if not explicitly mentioned, it maps to itself.



- For example:
 - $\text{date}(D, M, 2001)$ and $\text{date}(D1, \text{may}, Y1)$
 - Substitution: $e = [D/D1, M/\text{may}, Y1/2001]$

Applying substitution to clauses

- Substitution of a clause is defined by applying substitution to each of its literals:
$$e(p :- q_1, \dots, q_k) = e(p) :- e(q_1), \dots, e(q_k).$$
- Example:
C: $\text{pass_3401}(X) :- \text{student}(X, Y), \text{study_hard}(X).$
 $e = [X / \text{john}, Y / 3401]$
 $e(C) = \text{pass_3401}(\text{john}) :- \text{student}(\text{john}, 3401), \text{study_hard}(\text{john}).$

Applying substitution to literals

- Example:

$p(X, f(X, 2, Z), 5)$

$e = [X/5, Z/h(a, 2+X)]$

$e(p(X, f(X, 2, Z), 5)) = p(5, f(5, 2, h(a, 2+X)), 5)$

- Note:

- Simultaneous substitution
- X in $h(a, 2+X)$ is not substituted

- Example:

$r(X, Y)$

$e = [X/Y, Y/X]$

$e(r(X, Y)) = r(Y, X)$

- Example:

$r(X, f(2, Y))$

$e = [f(2, Y) / Z]$

illegal substitution- only variables can be substituted

Unifier

- Let p_1 and p_2 be two literals and let e be a substitution. We call e a unifier of p_1 and p_2 if $e(p_1)=e(p_2)$.
- Two literals are unifiable if such a unifier exists.
- Example:
 $p_1: \text{date}(D, M, 2001)$ and $p_2: \text{date}(D1, \text{may}, Y1)$
 $e=[D/15, D1/15, M/\text{may}, Y1/2001]$
 $e(p_1)= \text{date}(15, \text{may}, 2001)$
 $e(p_2)= \text{date}(15, \text{may}, 2001) \rightarrow e(p_1)=e(p_2)$
- e is a unifier of p_1 and p_2
- p_1 and p_2 are unifiable

Most General Unifier (mgu)

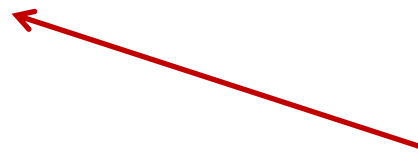
- A unifier e is said to be a most general unifier (mgu) of two literals/terms iff e is more general than any other unifier of the terms.

- Example:

$\text{date}(D, M, 2001)$ and $\text{date}(D1, \text{may}, Y1)$

$e_1 = [D/15, D1/15, M/\text{may}, Y1/2001]$: not the most general

$e_2 = [D1/D, M/\text{may}, Y1/2001]$

 **mgu**

Unification

- Called matching in Prolog
- Rules for matching two terms, S and T [Bratko]:
 - If S and T are constants, then S and T match only if they are the same object.
 - If S is a variable (and T is anything), then they match and S is substituted by T (*instantiated to T*). Conversely, if T is a variable, then T is substituted by S.
 - If S and T are structures, then they match iff
 - S and T have the same principal functor
 - All their corresponding components match

Examples

Are the following literals unifiable? What is their mgu?

1. $\text{triangle}(\text{point}(1,2), X, \text{point}(2,4))$ and $\text{triangle}(A, \text{point}(5, Y), \text{point}(2, B))$
unifiable: $\text{mgu}=[A/\text{point}(1,2), X/\text{point}(5,Y), B/4]$
2. $\text{horizontal}(\text{point}(1,X), Y)$ and $\text{vertical}(Z, A)$
not unifiable: $\text{horizontal} \neq \text{vertical}$
3. $\text{plus}(2,2)$ and 4
not unifiable
4. $\text{seg}(\text{point}(1,2), \text{point}(3,4))$ and $\text{seg}(f(1,2), Y)$
not unifiable: $\text{point} \neq f$

Unification vs. Matching

- Are $p(X)$ and $p(f(X))$ unifiable?
 $e=[X/f(X)]$
 $X=f(f(f(f(f(\dots ?!$
- This is not allowed in unification. Proper unification requires occurs check: a variable X can not be substituted by a term t if X occur in t .
- This is not done in Prolog's matching for efficiency reasons.
 - Therefore it is referred to as 'matching' in Prolog, and not 'unification'.

The resolution rule

- Given two clauses in the form:

$$A_0..A_i..A_m:-B_1...B_n \text{ and } C_1...C_k :- D_1..D_j..D_l$$

If e is a unifier of A_i and D_j (i.e. $e(A_i)=e(D_j)$)

Then the resolvent of the above two clauses is:

$$e(A_0).. e(A_{i-1})e(A_{i+1}).. e(A_m) e(C_1)..e(C_k) :- \\ e(B_1).. e(B_n) e(D_1).. e(D_{j-1})e(D_{j+1}).. e(D_l).$$

- Example:

$$C_1: p(Y):- r(X, Y), q(Y, Z).$$

$$C_2: :- p(f(1)).$$

Unifier of $p(f(1))$ and $p(Y)$: $e=[Y/f(1)]$

The resolvent of C_1 and C_2 : $:- r(X, f(1)), q(f(1), Z).$

Example

[Nilsson]

C0: proud(X) :- parent(X, Y), newborn(Y).

C1: parent(X, Y) :- father(X, Y).

C2: parent(X, Y) :- mother(X, Y).

C3: father(adam, mary).

C4: newborn(mary).

G0: :- proud(Z).

Unifier of proud(..) in C0 and G0: $e=[X/Z]$, resolvent:

G1: :- parent(Z, Y), newborn(Y).

Unifier of parent(..) in C1 and G1: $e=[X/Z, Y/Y]$, resolvent:

G2: :- father(Z, Y), newborn(Y).

To prevent mistakes, we rename the variables whenever we use a fresh copy of a clause.

Example (cont.)

[Nilsson]

G0: :- proud(Z).

(copy of) **C0:** proud(X_1) :- parent(X_1, Y_1), newborn(Y_1).

Resolve with G0: $e=[X_1/Z]$

G1: :- parent(Z, Y_1), newborn(Y_1).

(copy of) **C1:** parent(X_2, Y_2) :- father (X_2, Y_2).

Resolve with G1: $e=[X_2/Z, Y_2/Y_1]$

G2: :- father(Z, Y_1), newborn(Y_1).

(copy of) **C3:** father(adam, mary).

Resolve with G2: $e=[Z/adam, Y_1/mary]$

G3: :-newborn(mary).

(copy of) **C4:** newborn(mary).

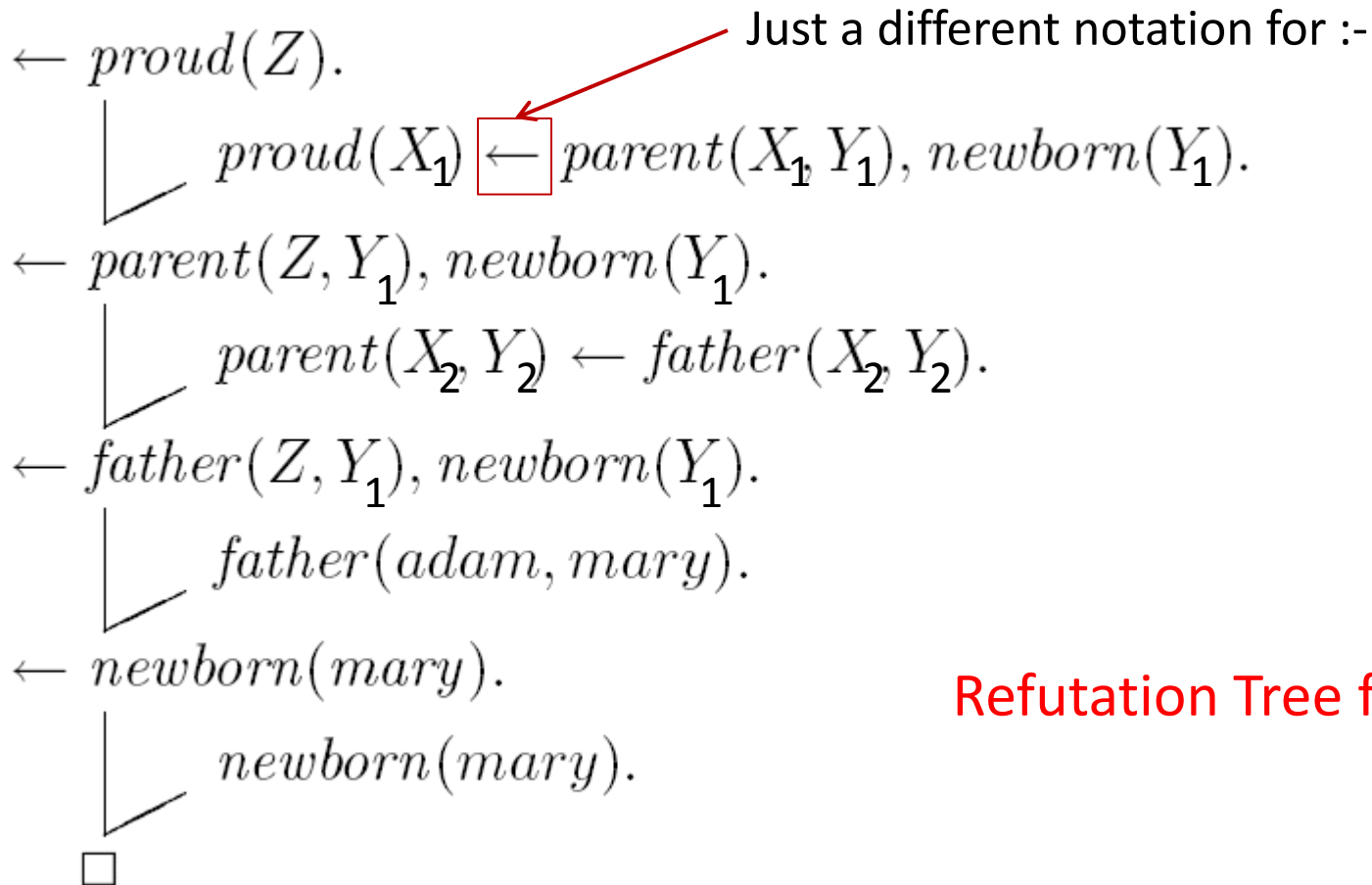
Resolve with G3: $e=[]$

G4: :-

Empty clause \rightarrow answer to query: true if Z=adam

Example (cont.)

[Nilsson]



Refutation Tree for G0

Linear Refutation

- We can resolve with different clauses and keep adding new clauses forever!
- To prevent this, Linear Refutation always starts with a goal (as the example showed previously).
- Prolog's computation rule:
Always selects the leftmost subgoal, although logically there is no order for the subgoals.
Example: When resolving **G1**: :- parent(Z,Y₁), newborn(Y₁)., parent(..) was selected to resolve upon.
Prolog also starts from the top of knowledge base and goes down the list of facts and rules.

Search Space

- Based on linear refutation and Prolog's computation rule, we know the search tree of Prolog.
- Search tree:
The root in the search tree is the main goal G_0 . A child node is a new goal G_i obtained through resolution. A link is labelled with the clause resolved with and the substitution.
- Example:
 - C0: grandfather(X,Z) :- father(X,Y), parent(Y,Z).
 - C1: parent(X,Y) :- father(X,Y).
 - C2: parent(X,Y) :- mother(X,Y).
 - C3: father(a,b):-.
 - C4: mother(b,c):-.
 - C5: mother(b,d):-.
 - G0: :- grandfather(a,X).**

Search Space (example)

← *grandfather(a, X).*

C0, [Z₀/X, X₀/a]

← *father(a, Y₀), parent(Y₀, X).*

C3, [Y₀/b]

← *parent(b, X).*

C1, [X₁/b, Y₁/X]

C2, [X₂/b, Y₂/X]

← *father(b, X).*

← *mother(b, X).*

Nothing to resolve with, backtrack!

C4, [X/c]

□ **TRUE, X=c ; (backtrack!)**

C5, [X/d]

□ **TRUE, X=d**

C0: grandfather(X,Z) :-
father(X,Y), parent(Y,Z).

C1: parent(X,Y) :- father(X,Y).

C2: parent(X,Y) :- mother(X,Y).

C3: father(a,b):-.

C4: mother(b,c):-.

C5: mother(b,d):-.

G0: :- grandfather(a,X).

Search Space

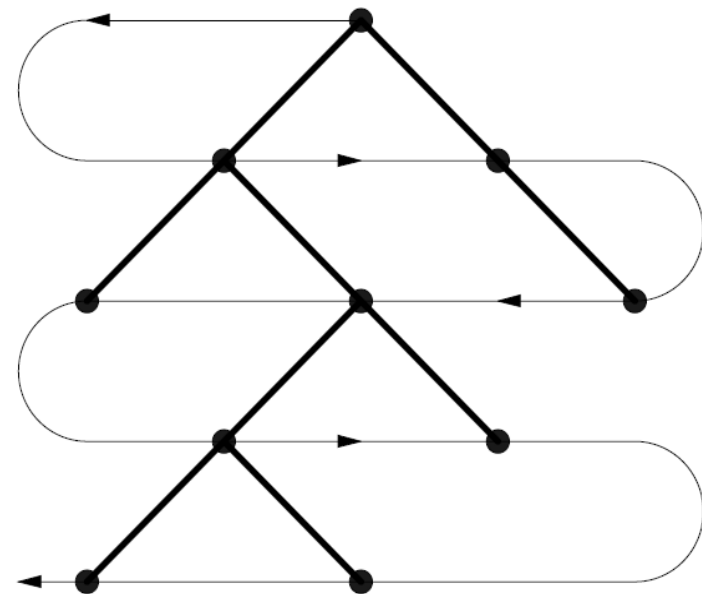
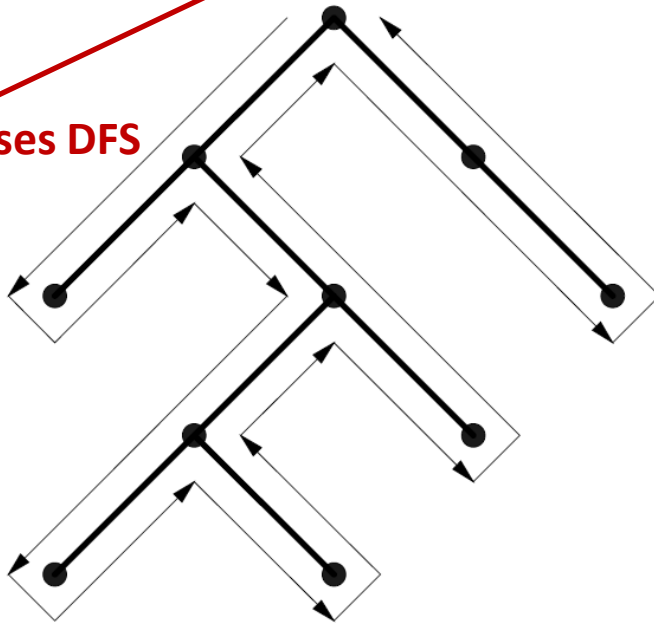
- What is the search strategy used by Prolog for searching the tree?

Depth First Search

or

Breadth First Search

Prolog uses DFS



Questions

1. Why is it ok to rename the variables every time we use a fresh copy of a clause (slide #24)? Doesn't that change the clause?!

Note that each variable X in our clause is actually bound by a universal quantifier ($\forall X$). In the last step of converting to CNF, we removed the quantifiers for simpler notation. Renaming the variable bound by a quantifier (dummy renaming) does not alter the semantics of the formula.

Questions

2. Why do we always have goals in each node of Prolog's search tree? Why don't we have facts and rules as resolvents?!

Each goal is a clause with no head and several literals in the body. On the other hand, each clause in the knowledge base is a fact or a rule, having exactly one literal in the head and zero or more literals in the body. Based on linear refutation, Prolog always resolves the goal with its set of facts and rules. The one literal in the head of the facts and rules, is always cut (resolved) with a subgoal, leaving no literals in the head. Therefore after each resolution step, the resolvent is always a goal.

Examples for resolution rule (Slide #23)

- Example 1: Consider two clauses:
student(john, 3401):-.
:- student(john, 3401).

We can resolve these two clauses upon the literal
“student(john,3401)”, since:

1. It is unifiable in both clauses after no substitution ($e=[]$).
2. It appears on the right side (of :-) in one clause and on the left side in the other clause.

The resolvent (cutting the literal in both clauses, and writing what is left from both clauses) is therefore:
:- (the empty clause)

Examples for resolution rule (Slide #23)

- Example 2: Consider two clauses:
student(john, 3401):-.
:- student(X, 3401).

We can resolve these two clauses upon the literals
“student(john,3401)” and “student(X,3401)”, since:

1. These literals are unifiable after substitution $e=[X/\text{john}]$.
 $e(\text{student}(\text{john},3401))= e(\text{student}(X,3401))$
2. One appears on the right side (of :-) in one clause and the other on the left side in the other clause.

The resolvent (cutting the literal in both clauses, and writing what is left from both clauses) is therefore:

:- (the empty clause)

Examples for resolution rule (Slide #23)

- Example 3: Consider two clauses:

$\text{pass}(X) :- \text{student}(X, 3401), \text{study}(X).$

$:- \text{pass}(\text{john}).$

We can resolve these two clauses upon the literals

“ $\text{pass}(\text{john})$ ” and “ $\text{pass}(X)$ ”, since:

1. These literals are unifiable after substitution $e=[X/\text{john}]$.
 $e(\text{pass}(\text{john})) = e(\text{pass}(X))$
2. One appears on the right side (of $:-$) in one clause and the other on the left side in the other clause.

The resolvent (cutting the literal in both clauses, and writing what is left from both clauses) is therefore:

$:- e(\text{student}(X, 3401)), e(\text{study}(X)).$

After applying the substitution, it will simplify to:

$:- \text{student}(\text{john}, 3401), \text{study}(\text{john}).$