ECE 479/579
Principles of Artificial Intelligence – Part II
Spring 2005

Dr. Michael Marefat (marefat@ece.arizona.edu)

Unification

Finding substitutions for terms is called unification.

$W1(?x) \Rightarrow W2(?x)$

W (A) A / ?x

A Substitution instance is an instance of an expression obtained by putting terms for variables.

We can substitute constants for variables, but not variables for constants.

Example:

Child_of [?x , brother (?y) , Janet]

Instance 1: Child_of [?z, brother(?w), Janet]

alphabetic variant.

Instance 2: Child_of [?x, brother (Jerry), Janet]

Instance 3: Child_of [driver_of(z3), brother(Jerry), Janet]

Instance 4: Child_of [Jamie, brother(Jerry), Janet] Ground Instance.

<<< Refer to Section 9.2 of the text >>>

Substitutions are represented as sets of ordered pairs:

S= { t_1/v_1, t_2/v_2,……., t_n/v_n}

where t_i / v_i means that term t_i is to be substituted for variable v_i throughout the expression.

Rules governing substitutions:

1. Each occurrence of the variable v_i must be substituted with the same term t_i .

2. No variable can be substituted by a function containing the same variable.

{ g(?x) / ?x } is not allowed.

Substitutions in above example:

S1 = { z / ?x , w / ?y }

S2 = { Jerry / ?y }

S3 = { driver_of (z3) / ?x , Jerry / ?y }

S4 = { Jamie / ?x , Jerry / ?y }

If we have an expression W and we have a substitution S, then WS represents applying S to W.

Child_of [?z, brother (?w), Janet] =

Child_of [?x, brother (?y), Janet] s1

Properties of Substitutions.

- Composition:

Compositions of substitutions S1, S2 is denoted by S1S2.

To compute S1S2 do:

(1) Apply all substitutions in S2 to the terms in S1.

(2) Add any pairs in S2 which don't have variables among variables in S1.

Example:

S1 = { g(?x, ?y) / ?z }

S2 = { A/?x, B/?y , C/ ?w, D/ ?z }

S1S2 = {g(?x, ?y) / ?z } { A/?x, B/?y , C/ ?w, D/ ?z }

= { g (A, B)/?z , A/ ?x, B/ ?y, C/ ?w }

Automatic Deduction

W1(A)

W1(?x) ⇒ W2(?x)

W2(?x) ⇒ W3(?x)

……..

Chaining mechanisms

- Forward Chaining
  -start with facts and go forward
- Backward Chaining
  -start with goals and compute backward to facts.

Forward Chaining

We have a database consisting of facts and rules (implications). Implications are used as forward rules, continue this application of rules repetitively until we find the goal, or until no more rule applications can be made.

<<< Refer to Section 9.3 of the text >>>

Computational steps:

1. Convert fact expressions into AND/OR form.

k-connectors in AND/OR graph represent disjunctions and 1-connectors represent conjunctions.

2. Convert rules to appropriate form.

3. Apply rules to the AND/OR graph, search for solutions of AND/OR graph, and see if they match out goal.

Forward systems can deduce goals that are represented by disjunctions of literals.

When do we stop?

We stop when we find a solution graph all leaves of which match with goal.

Rules in forward chaining have the form:

$L \Rightarrow W$ ← Any wff

Single literal.

if we have

$L1 \vee L2 \Rightarrow W$

can always be broken into

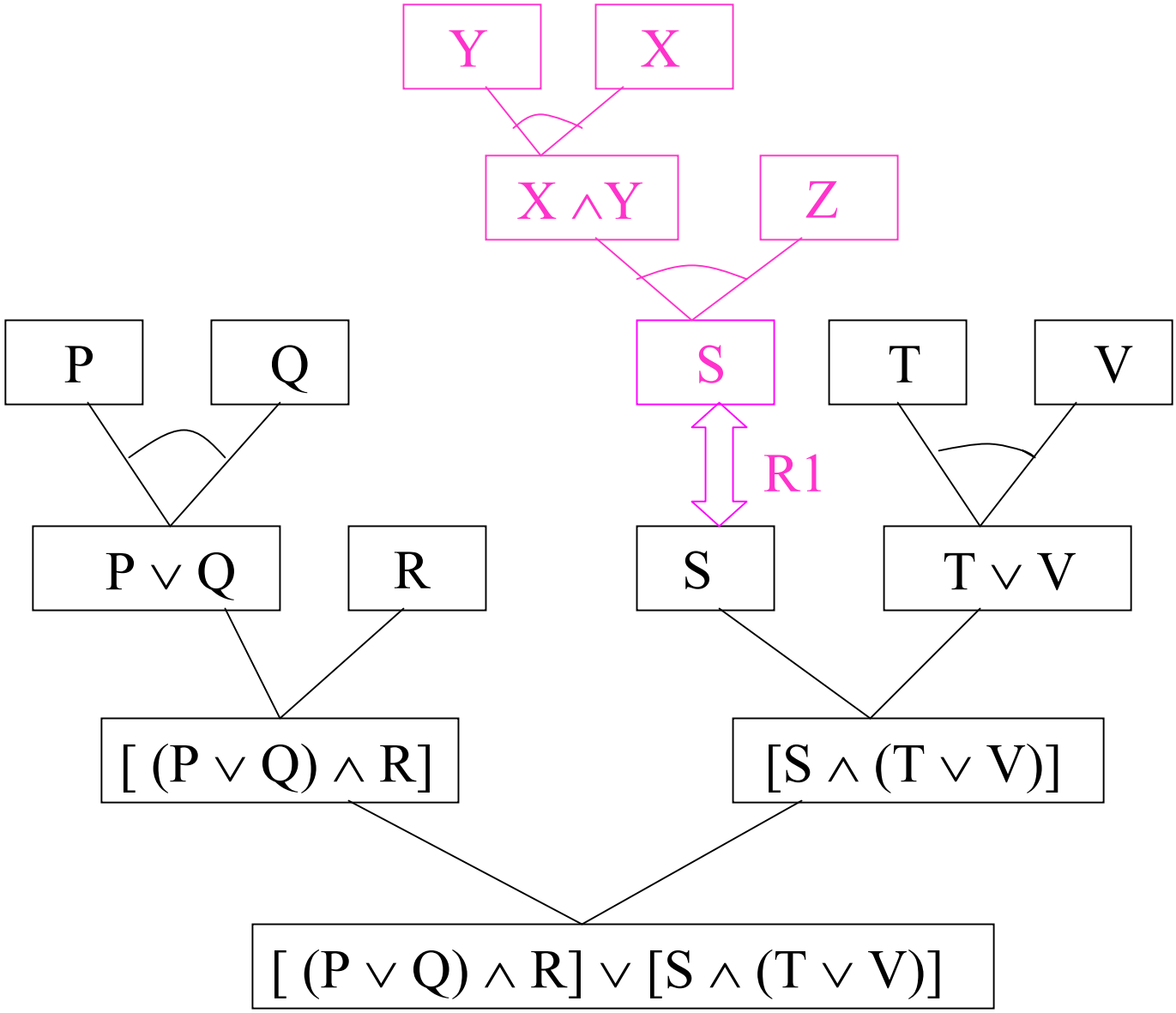
$L1 \Rightarrow W$ (R1)

$L2 \Rightarrow W$ (R2)

Ex: Forward deduction without variables.

Fact: $[(P \vee Q) \wedge R] \vee [S \wedge (T \vee V)]$

Rule: $S \Rightarrow (X \wedge (Y \vee Z)$



Original Solution Graphs:

$R \vee S$	Clause 1
$P \vee Q \vee S$	Clause 2
$T \vee V \vee R$	Clause 3
$P \vee Q \vee T \vee V$	Clause 4

New Solution Graphs:

$R \vee Z \vee X$

$R \vee Z \vee Y$

$P \vee Q \vee Z \vee X$

$P \vee Q \vee Z \vee Y$

What if more than one rule ? the same way as before.

Ex:

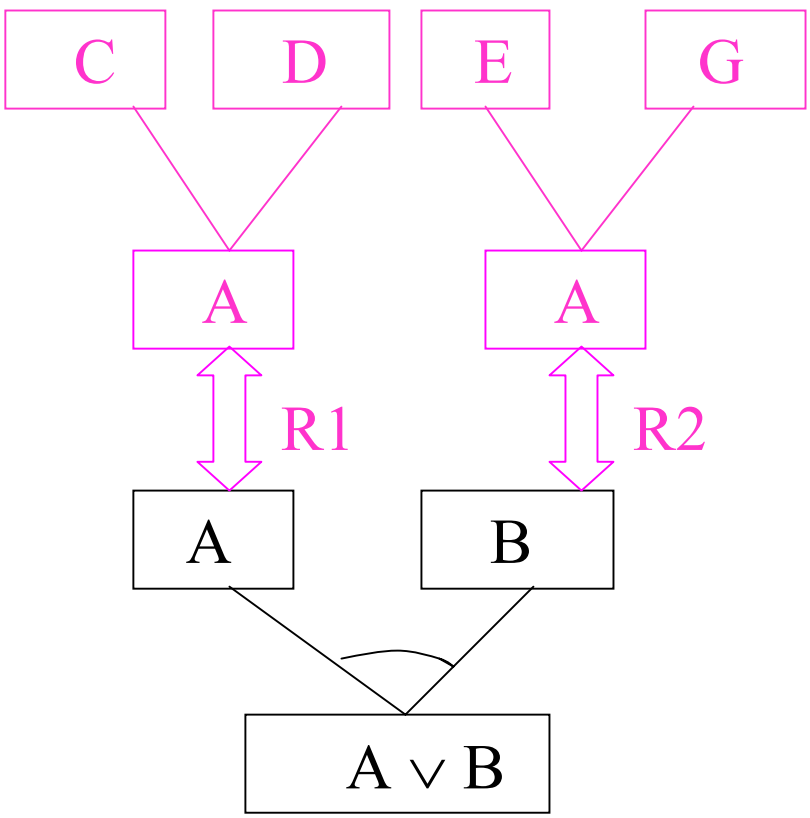
Fact: $A \vee B$

Rules:

R1: $A \Rightarrow C \wedge D$

R2: $B \Rightarrow E \wedge G$

Goal: $C \vee G$



Solution graphs:

1. $C \vee E$
2. $C \vee G$
3. $D \vee E$
4. $D \vee G$

Forward chaining with variables.

a. No Quantifiers:

to apply rules form substitutions.

Only substitutions which are consistent can be applied simultaneously.

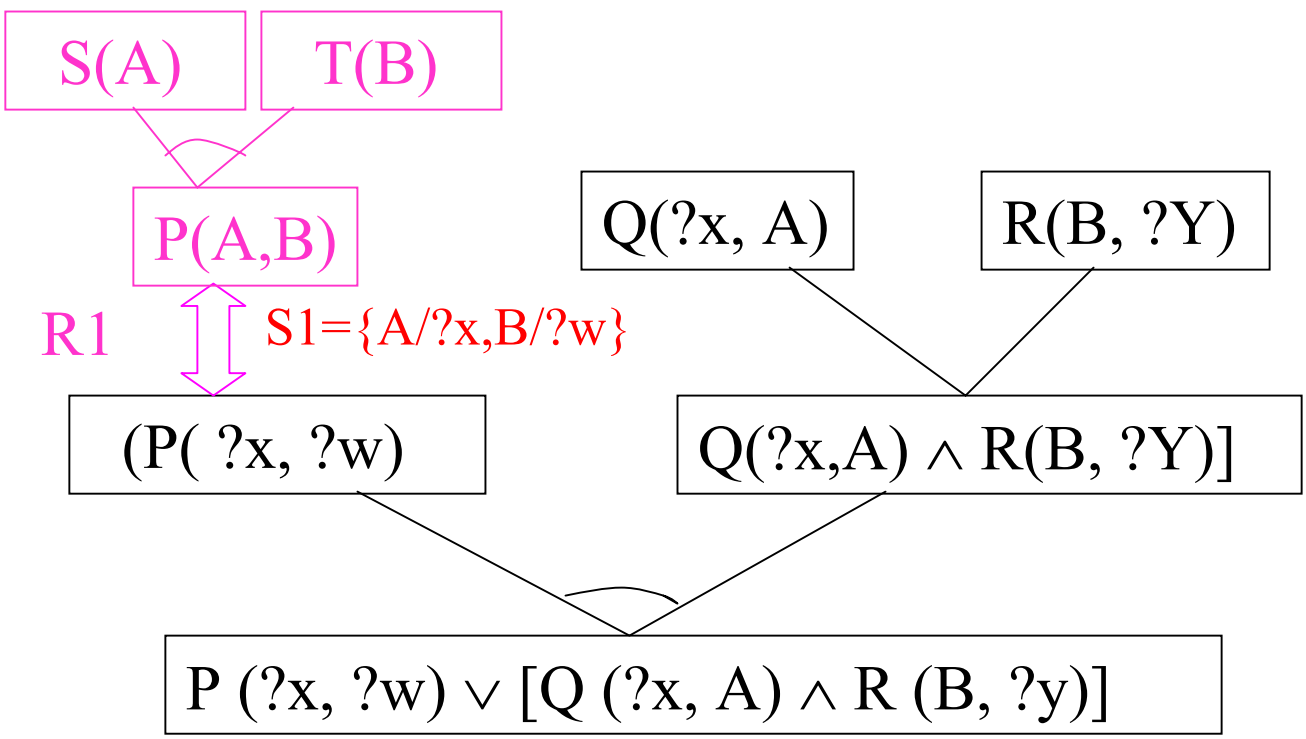
Ex:

Fact:

$$P(?x, ?w) \vee [\, Q\,(\, ?x, A) \wedge R(\, B, ?y)]$$

Rules:

$$R1: P(A, B) \Rightarrow [S(A) \vee T(B)\,]$$



Solution Graphs:

1. [S(A) $\vee$ T(B) $\vee$ Q (?x, A)] S1 = { A/?x, B/?w)

W1

= [S(A) $\vee$ T(B) $\vee$ Q (A, A)]

2. [S(A) $\vee$ T(B) $\vee$ R (B, ?y)] S1

= [S(A) $\vee$ T(B) $\vee$ R(B, ?y)]

Ex 2: More than one rule:

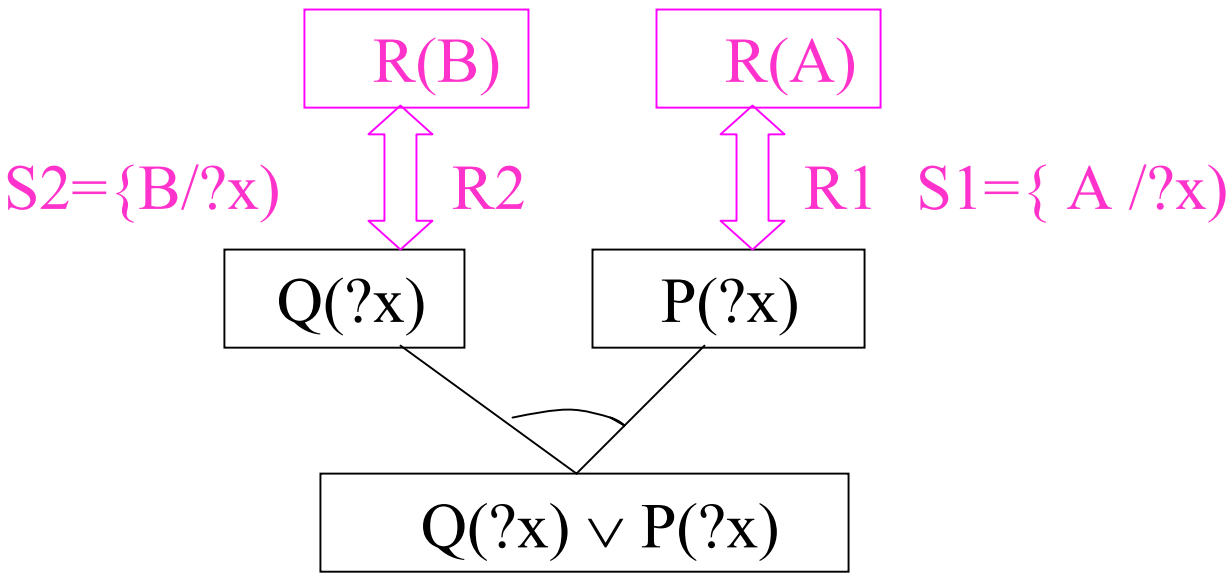
fact: $Q(?x) \vee P(?x)$

Rules:

$R1: P(A) \Rightarrow R(A)$

$R2: Q(B) \Rightarrow R(B)$

Generate all clauses obtained from Solution Graphs after all possible inferences.



Solution Graphs

1. R(B) $\vee$ R(A) if S1 and S2 are consistent.

   S1={A/ ?x} S2 = { B/ ?x}

   S1 and S2 are not consistent.

because they substitute different entities for the same variable ?x.

How to determine if two substitutions S1, S2 are consistent?

To do this form two sets T, V where

T = { t_1, t_2,…… t_n} are terms from both S1 and S2.

V = { v_1, v_2,…… v_n} are variables from both S1 and S2.

Then assign t_1 to v_1 and propagate through out the V set. At any point there is more than one constant to be assigned to a variable, the substitutions are inconsistent.

Ex: S1={A/ ?x} S2 = { B/ ?x}

T= { A B }

V = { ?x ~~?x~~}

A

Expressions with Quantified variables.

Do the same as before, but: Preprocess facts and rules.

To preprocess facts and rules:

1. Remove implications.

2. Reduce the scope of negations.

3. Skolemize existential quantifiers.

4. Drop Universal Quantifiers.

5. Rename Variables.

- If processing a rule, restore implication.

Ex:

$(\exists\ ?u)\ (\forall\ ?v)\ \{\ Q\ (?v, ?u) \wedge$

$\neg[\ [R\ (\ ?v) \vee P(\ ?v)] \wedge S\ (?u, ?v)]\ \}$

1. No implications $\rightarrow$ Nothing to do.

2. $(\exists\ ?u)\ (\forall\ ?v)\ \{\ Q\ (?v, ?u) \wedge$

$[\ [\neg\ R\ (\ ?v) \wedge \neg\ P(\ ?v)] \vee \neg\ S\ (?u, ?v)]\ \}$

3. Skolemize existential quantifiers.

Rules for skolemization:

1. If the quantifier for existential variable falls outside the scope of all Universal quantifiers, replace it with a constant such as A, B, ……

2. If the quantifier for existential variable falls within the scope of K universal quantifiers, then replace it with a function of those K variables, such as f (?x, ?y), or g(?z).

$(\forall\ ?v)\ \{\ Q\ (?v, A) \wedge$

$[\ [\neg\ R\ (\ ?v) \wedge \neg\ P(\ ?v)] \vee \neg\ S\ (A, ?v)]\ \}$

4. Drop Universal Quantifiers:

Q (?v, A) ∧

[[¬ R (?v) ∧ ¬ P(?v)] ∨ ¬ S (A, ?v)]

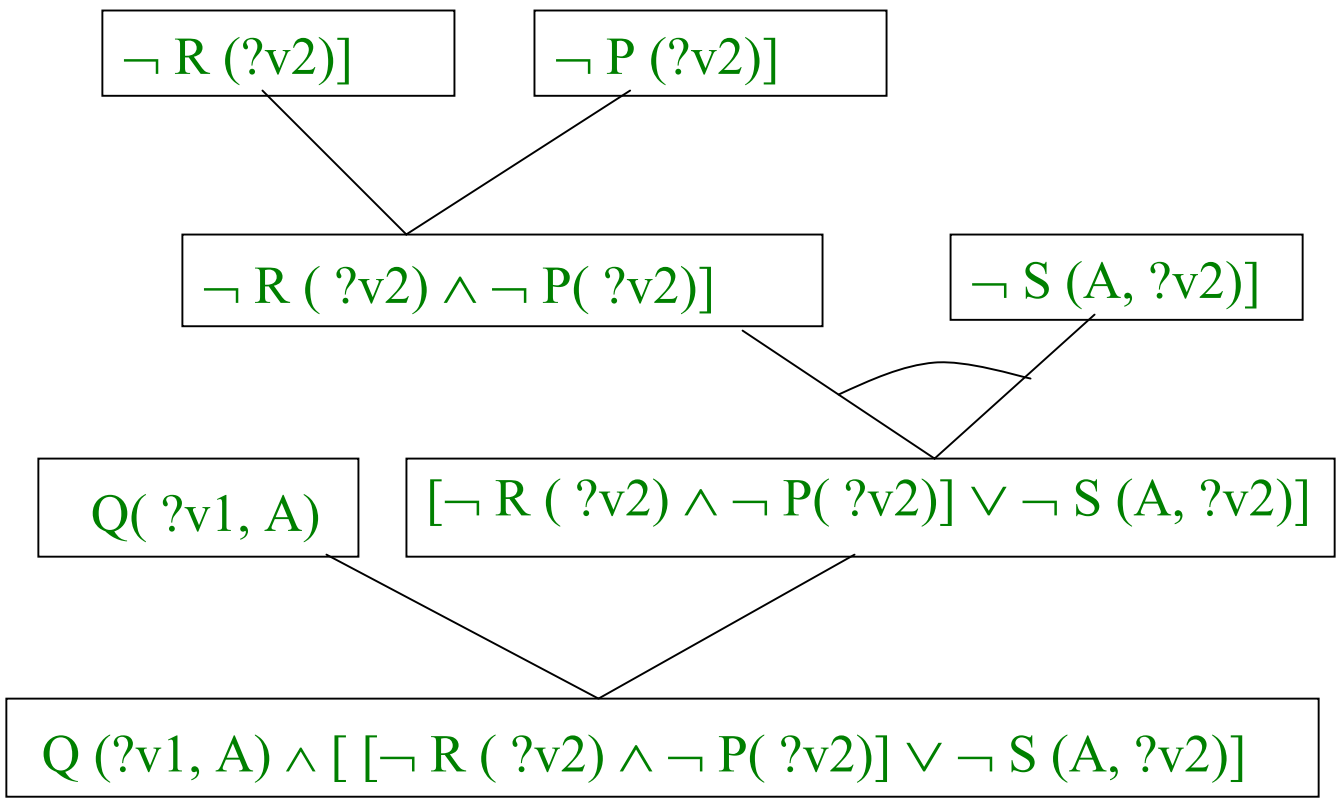
5. Rename variables.

at the outermost conjunctions, rename variables so that they are unique to each conjunction.

Q (?v1, A) ∧

[[¬ R (?v2) ∧ ¬ P(?v2)] ∨ ¬ S (A, ?v2)]

And/or graph for this wff.



An example Rule:

$(\forall\ ?x)\{[(\exists\ ?y)\ (\forall\ ?z)\ P(?x, ?y, ?z)] \Rightarrow (\forall\ ?u)\ \ Q(\ ?x, ?u)\}$

1. Remove implications. $W1 \Rightarrow W2$

$\neg W1 \vee W2$

$(\forall\ ?x)\{\neg\ \ (\exists\ ?y)\ (\forall\ ?z)\ P(?x, ?y, ?z)\ \ \vee (\forall\ ?u)\ \ Q(\ ?x, ?u)\}$

2. Reduce scope of negations.

$(\forall\ ?x)\{(\forall\ ?y)\ (\exists\ ?z) \neg\ P(?x, ?y, ?z)\ \ \vee (\forall\ ?u)\ \ Q(\ ?x, ?u)\}$

3. Skolemize existential

$(\forall\ ?x)\{\forall\ ?y) \neg\ P(?x, ?y, f(?x, ?y))\ \ \vee (\forall\ ?u)\ \ Q(\ ?x, ?u)\}$

4. Drop Universals

$\neg\ P(?x, ?y, f(?x, ?y))\ \ \vee Q(\ ?x, ?u)\}$

5. Rename Variables.

Not needed because no conjunctions.

This is a rule, need to restore implication:

$P(?x, ?y, f(?x, ?y\ )) \Rightarrow Q(\ ?x, ?u)\}$

Ex. of a complete forward System

Fido barks and bites, or Fido is not dog. All terriers are dogs. Anyone who barks is noisy.

Find someone who is not a terrier and who is noisy.

Steps:

1. Model given info. with facts and rules in FOPL.

2. Process facts and rules to appropriate form.

3. Construct and/or graph, apply rules, extract consistent solution graphs.

Ontology:

Barks (?x) ← ?x Barks

Bites (?x) ← ?x Bites

Dog (?x) ← ?x is a dog.

Terrier (?x) ← ?x is a terrier

Noisy (?x) ← ?x is noisy.

Fido ← a constant.

Facts:

$[\text{Barks(Fido)} \wedge \text{Bites (Fido)}] \vee \neg \text{Dog (Fido)}$

Rules:

R1: $(\forall\ ?x)\ [\neg \text{Dog}\ (?x) \Rightarrow \neg \text{Terrier}\ (?x)]$

R2: $(\forall\ ?y)\ [\ \text{Barks}\ \ (?x) \Rightarrow\ \text{Noisy}\ (?y)]$

Goal: $\neg$ Terrier (?z) $(\forall ?x) \vee$ Noisy (?z)

Preprocessing:

- Fact does not need processing $\rightarrow$no quantifiers

- Rules need processing.

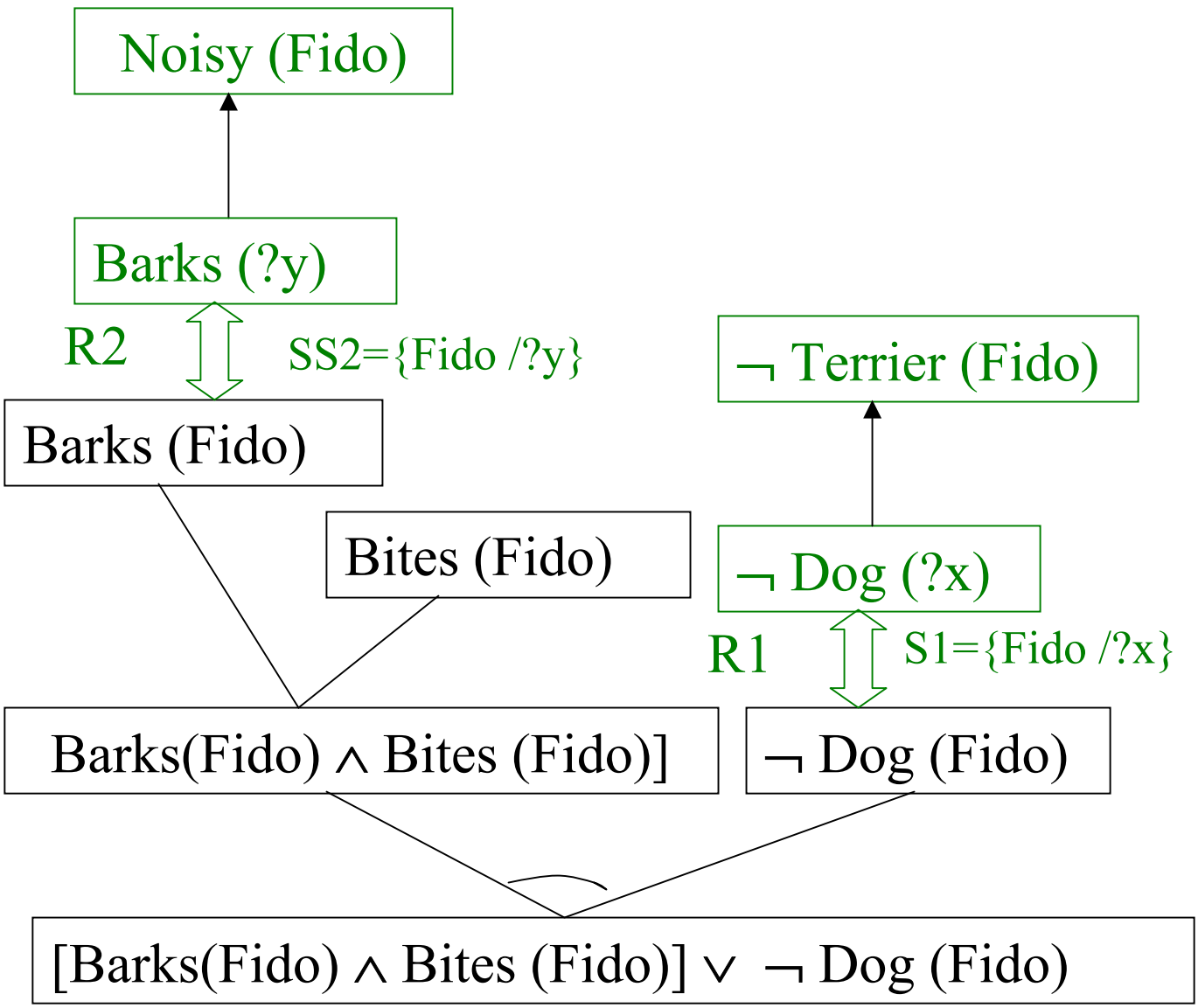
R1: $(\forall\ ?x)\ [\ \text{Dog}\ (?x) \vee \neg \text{Terrier}\ (?x)]$

$\text{Dog}\ (?x) \vee \neg \text{Terrier}\ (?x)$

$\neg \text{Dog}\ (?x) \Rightarrow \text{Terrier}\ (?x)$

R2: $\text{Barks}\ (?y) \Rightarrow \text{Noisy}\ (?y)$

Form AND/OR graph and apply rules.



Solution Graphs:

- After the first deduction (R1):

[¬ Terrier (Fido) ∨ Bites (Fido)] S1
[¬ Terrier (Fido) ∨ Barks (Fido)] S1
where S1 ={Fido/ ?x}

- After the second deduction (R2):
[¬ Terrier (Fido) ∨ Noisy (Fido)] S1,S2 combined

T ={ Fido Fido}
V={ ?x ?y}

Goal: ¬ Terrier (?z) ∨ Noisy (?z)

Unify S={ Fido/?z}

Backward Chaining in Rule Systems

Goals are conjunctions of predicates.

Process as follows:

1. Skolemize Universal variables instead of existentially quantified variables, Existential quantifications get dropped.

2. Express goal as AND/OR graph.

3. Rules must be of the form $W \Rightarrow L$ (not $L \Rightarrow W$)

4. K connectors represent conjunctions.

(not disjunctions)

5. Start with matching rules to subgoals, and stop when there is a solution graph unifying with facts (consistently).

<<< Refer to Section 9.4 of the text >>>

Ex: Express goals as AND/OR graph:

$$(\exists\ ?y)\ (\forall\ ?x)\ \{\ P\ (?x) \Rightarrow [Q(\ ?x,\ ?y) \wedge \neg[\ R\ (\ ?x) \wedge S\ (?y)]]\ \}$$

1. Remove Implications:

$$(\exists\ ?y)\ (\forall\ ?x)\ \{\neg\ P\ (?x) \vee [Q(\ ?x,\ ?y) \wedge \neg[\ R\ (\ ?x) \wedge S\ (?y)]]\}$$

2. Reduce scope of negations:

$$(\exists\ ?y)\ (\forall\ ?x)\ \{\neg\ P\ (?x) \vee [Q(\ ?x,\ ?y) \wedge [\neg\ R\ (\ ?x) \vee \neg\ S\ (?y)]]\}$$

3. Skolemize Universals.

$$(\exists\ ?y)\ \{\neg\ P\ (f(?y)) \vee [Q(\ f(?y),\ ?y) \wedge [\neg\ R\ (\ f(?y)) \vee \neg\ S\ (?y)]]\}$$

4. Drop Existential Quantifiers

$$\neg\ P\ (f(?y)) \vee [Q(\ f(?y),\ ?y) \wedge [\neg\ R\ (\ f(?y)) \vee \neg\ S\ (?y)]]$$

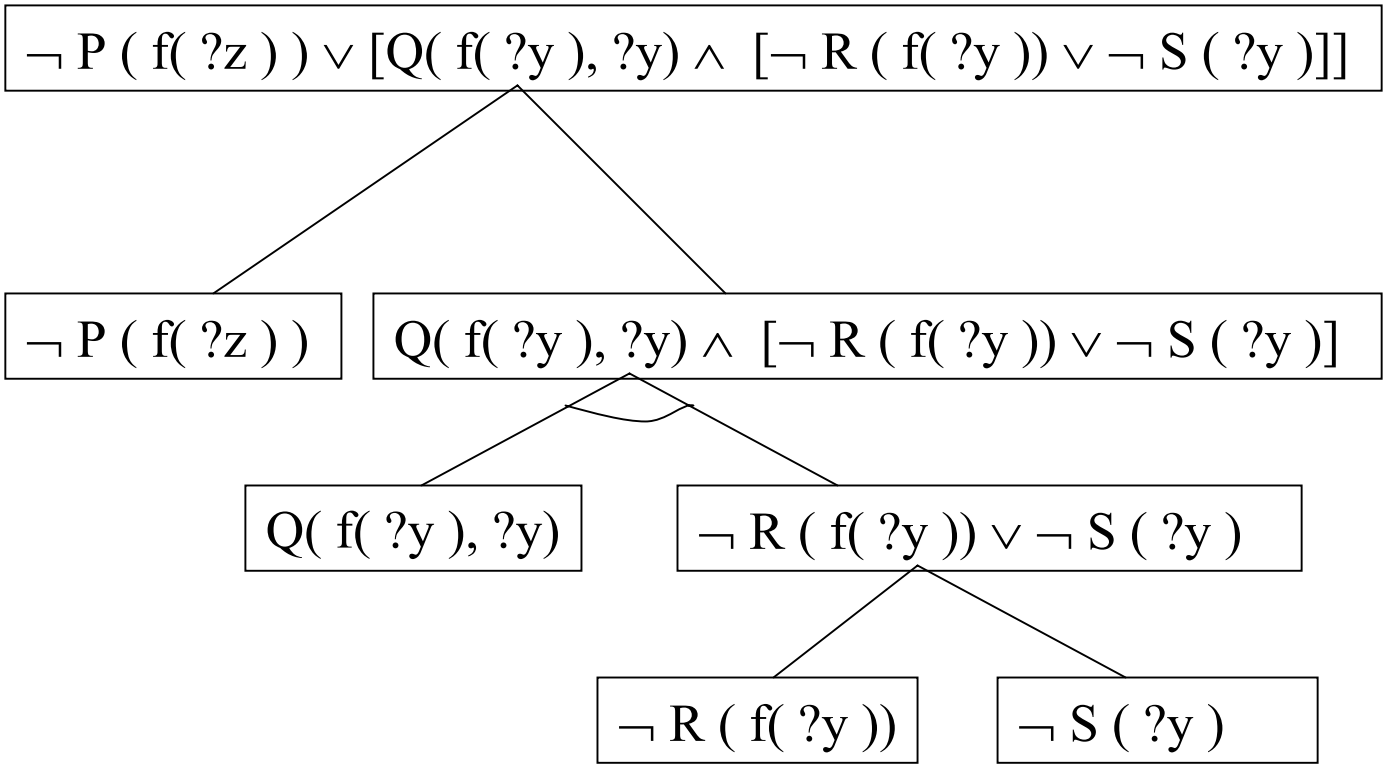
5. Rename Variables:

Look at outermost layer, if disjunctions of different subexpressions, use different unique variables for each subexpression.

$\neg P(f(?z)) \vee [Q(f(?y), ?y) \wedge$

$[\neg R(f(?y)) \vee \neg S(?y)]]$

Construct AND/OR Graph:

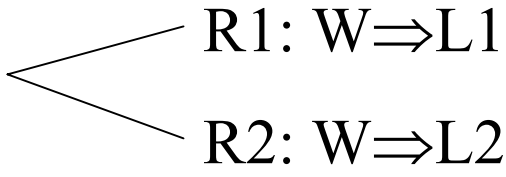


Solution Graphs:

1. $\neg P(f(?z))$
2. $Q(f(?y), ?y) \wedge \neg R(f(?y))$
3. $Q(f(?y), ?y) \wedge \neg S(?y)$

A Point about backward rules:

rules in B.C. must be $W \Rightarrow L$

if we have $W \Rightarrow L1 \wedge L2$ 

Ex:

facts and rules { Dogs and cats are animals. Fido is a dog and Myrtle meows. If a dog wags its tail, it is friendly. Anyone who meows is a cat. If something is friendly and does not bark, then other animals are not afraid of it, Suppose Fido wags its tail and does not bark.

Goal { Find a cat and a dog such that the cat is not afraid of the dog.

Model symbolically in FOPL.

Ontology:

Dog(?x) → ?x is a dog.

Animal(?x) → ?x is an animal

Cat (?x) → ?x is a cat.

Wagstail (?x) → ?x wags its tail.

Bark(?x) → ?x Barks.

Meows (?x) → ?x Meows

Friendly (?x) → ?x is friendly.

Afraid(?x, ?y) → ?x is afraid of ?y.

Facts

F1: Dog(Fido)

F2: Meows(Myrtle)

F3: Wagstail (Fido)

F4: ¬ Bark (Fido)

Rules:

R1: Dog(?x3) ⇒ Animal (?x3)

R2: Cat (?x4) ⇒ Animal (?x4)

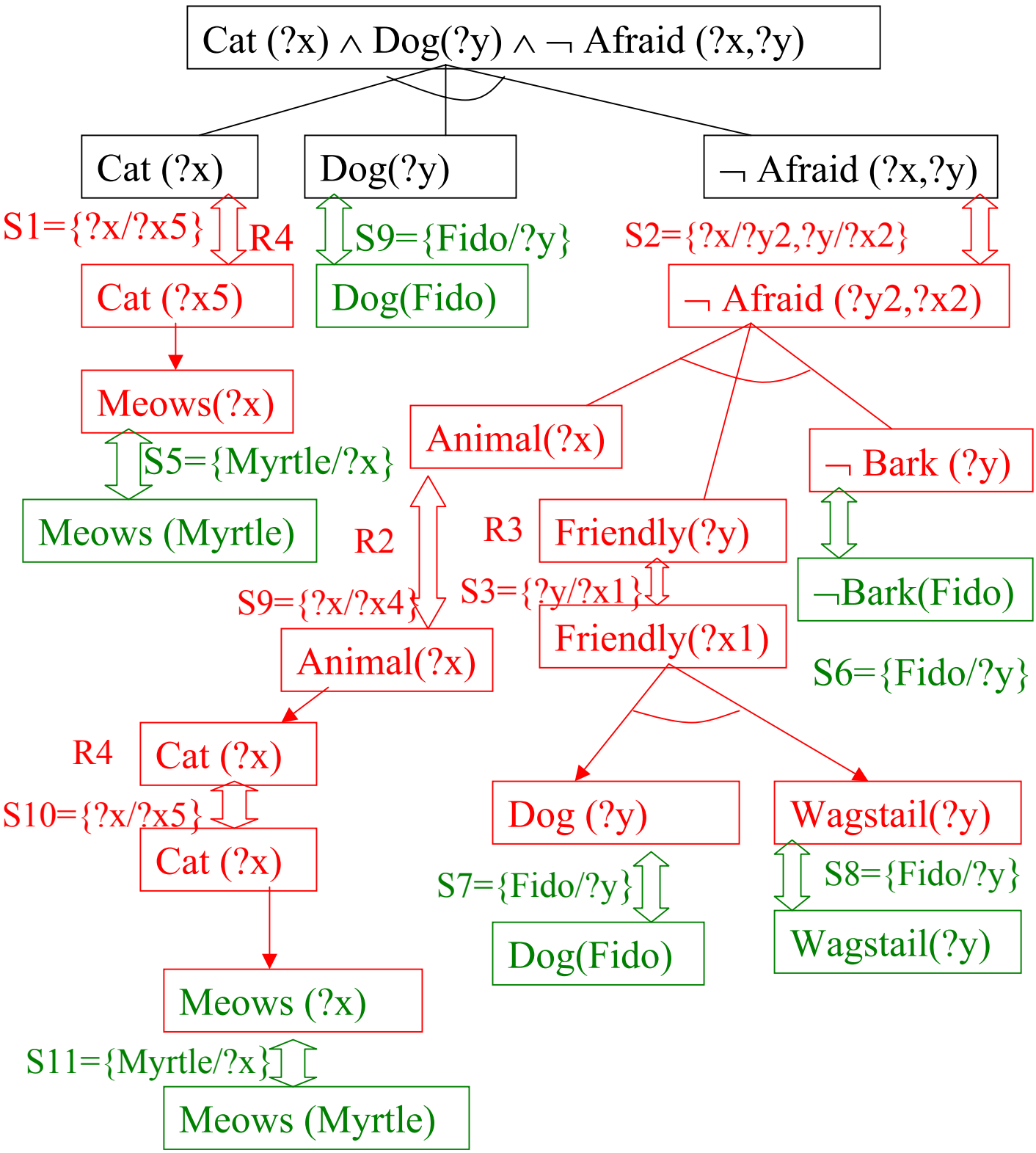
R3: [Dog (?x1) ∧ Wagstail (?x1)]⇒Friendly(?x1)

R4: Meows (?x5) ⇒ cat (?x5)

R5: [Friendly (?x2) ∧ ¬ Bark (?x2) ∧ Animal (?y2)] ⇒ ¬ Afraid (?y2, ?x2)

Goal:

$(\exists ?x)(\exists ?y)[Cat(?x) \wedge Dog(?y) \wedge \neg Afraid(?x,?y)]$



Consistency of substitutions:

T={ ?x ?x ?y ?y Fido Myrtle Fido}

V={?x5 ?y2 ?x2 ?x1 ?y ?x ?y }

T={ Fido Fido ?x ?x Myrtle }

V={ ?y ?y ?x4 ?x5 ?x }

Fido/?y Fido/?x1 Fido/?x2

Myrtle/?x Myrtle/?x5 Myrtle/?y2

Myrtle/?x4

Automatic Deduction using Resolution.

1. A set of facts.

Convert them to clauses (Clauses contain only disjunctions.

2. A goal to prove

Negate it and Convert it into clauses.

3. Apply resolution until a NIL expression is generated.

Resolution

I. Ground Clauses (no variables)

Suppose we have two clauses:

$P_1 \vee P_2 \vee \ldots\ldots P_n$ Clause 1

$\neg P_1 \vee Q_2 \vee Q_3 \vee \ldots\ldots \vee Q_m$ Clause 2

From these two parent clauses we can generate a new clause:

$P_2 \vee \ldots\ldots P_n \vee Q_2 \vee Q_3 \vee \ldots\ldots\ldots\ldots\ldots \vee Q_m$

This new clause is the resolvent of the parents.

<<< Refer to Section 9.5 of the text >>>

II. Clauses with Variables

Let's represent the parent clauses as two sets $\{L_i\}$ and $\{M_i\}$. There is a disjunction between elements

of each set. Suppose $\{l_i\}$ and $\{m_i\}$ are subsets of $\{L_i\}$ and $\{M_i\}$ such that there is mgu, S, that makes the sets $\{l_i\}$ and $\{m_i\}$ complementary. Then a resolvent of $\{L_i\}$ and $\{M_i\}$will be

$$\{ \{L_i\} - \{l_i\}\}S \cup \{\{M_i\} - \{m_i\}\}S$$

Ex: Consider two clauses:

$$P[?x, f(A)] \vee P[?x, f(?y)] \vee Q(?y)$$

$$\neg P[?x, f(A)] \vee Q(?z)$$

$$\{L_i\} = \{P[?x, f(A)] , P[?x, f(?y)] , Q(?y)\}$$

$$\{M_i\} = \{\neg P[?x, f(A)] , Q(?z)\}$$

Let

$$\{l_i\} = \{P[?x, f(A)]\}$$

$$\{m_i\} = \{\neg P[?x, f(A)] \}$$

$$S = \{ ?x/ ?z\}$$

The resolvent will be:

$\{ P[?x, f(?y)] , Q(?y) \}_S \cup \{\neg Q(?z) \}_S$

$S = \{ ?x/?z\}$

$= \{ P[?x, f(?y)] , Q(?y) \} \cup \{\neg Q(?x) \}$

resolvent = $P[?x, f(?y)] \vee Q(?y) \vee \neg Q(?x)$

Two clauses may have more than one resolvent, Because there may be more than one way to chose $\{l_i\}$ and $\{m_i\}$.

If

$\{l_i\} = \{P[?x, f(?y)]\}$

$\{m_i\} = \{\neg P[?z, f(A)] \}$ $S = \{ ?z/?x, A/?y\}$

resolvent:

$P[?z, f(A)] \vee Q(A) \vee \neg Q(?z)$

We can have more than one element in the subsets $\{l_i\}$, $\{m_i\}$.

If $\{l_i\} = \{P[?x, F(A)], P[?x, f(?y)]\}$

$\{m_i\} = \{\neg P[?z, f(A)]\}$

$S = \{ ?z/ ?x, A/ ?y \}$

the new resolvent:

$\{ Q (?y)\}\ S \cup \{\neg Q (?z)\}\ S$

$Q(A) \vee \neg Q (?z)$

Converting wffs into clauses

9 step process:

take as example this wff:

$(\forall\ ?x)\ \{\ P\ (?x) \Rightarrow \{(\forall\ ?y)\ [P(\ ?y) \Rightarrow\ P[f(?x,\ ?y)]] \wedge$

$\neg\ (\forall\ ?y)\ [\ Q(\ ?x,\ ?y) \Rightarrow P\ (?y)]\}\}$

(1) Eliminate implications: $W1 \Rightarrow W2$

$\neg\ W1 \vee\ W2$

$(\forall\ ?x)\ \{\neg\ P\ (?x) \vee \{(\forall\ ?y)\ [\neg\ P(\ ?y) \vee P[f(?x,\ ?y)]] \wedge$

$\neg\ (\forall\ ?y)\ [\neg\ Q(\ ?x,\ ?y) \vee P\ (?y)]\}\}$

(2) Reduce scope of negations:

$(\forall\ ?x)\ \{\neg\ P\ (?x) \vee \{(\forall\ ?y)\ [\neg\ P(\ ?y) \vee P[f(?x,\ ?y)]] \wedge$

$\neg\ (\exists\ ?y)\ [Q(\ ?x,\ ?y) \vee \neg\ P\ (?y)]\}\}$

(3) Standardize variables: rename variables to make sure every quantifier has its own distinct variable.

$(\forall\ ?x)\ \{\neg\ P\ (?x) \vee \{(\forall\ ?y)\ [\neg\ P(\ ?y) \vee P[f(?x,\ ?y)]] \wedge$

$\neg\ (\exists\ ?w)\ [Q(\ ?x,\ ?w) \vee \neg\ P\ (?w)]\}\}$

(4) Skolemize existential quantifiers.

(replace existential variables with constants if they are not in scope of any universal quantifier, or a function of the universal variables in whose scope they appear).

$(\forall\ ?x)\ \{\neg\ P\ (?x) \vee \{(\forall\ ?y)\ [\neg\ P(\ ?y) \vee P[f(?x,\ ?y)]] \wedge$

$[Q(\ ?x,\ g(?x)) \vee \neg\ P\ (g(?x))\ ]\}\}$

(5) Convert to prenex: move all Universal quantifiers to the front of the wff.

$(\forall\ ?x)\ (\forall\ ?y)\ \{\neg\ P\ (?x) \vee \{[\neg\ P(\ ?y) \vee P[f(?x,\ ?y)]] \wedge$

$[Q(\ ?x,\ g(?x)) \vee \neg\ P\ (g(?x))\ ]\}\}$

(6) Put in conjunctive normal form.

Use distributive rules on $\vee$ and $\wedge$ until the matrix becomes conjunctions of disjunctions.

$W1 \vee (W2 \wedge W3) \longrightarrow$

$(W1 \vee W2)\ \ \wedge\ \ (\ W1 \vee\ W3)$

(7) Eliminate Universal Quantifiers

$(\forall\ ?x)\ (\forall\ ?y)\ \{[\neg\ P\ (?x) \vee \neg\ P(\ ?y) \vee P[f(?x,\ ?y)]] \wedge$

$[\neg\ P\ (?x) \vee Q(\ ?x,\ g(?x))] \wedge [\neg\ P\ (?x) \vee \neg\ P\ (g(?x))\ ]\}\}$

(8) Eliminate $\wedge$ Symbols to get the initial set of clauses.

$\neg\ P\ (?x) \vee \neg\ P(\ ?y) \vee P[f(?x,\ ?y)]]$

$\neg\ P\ (?x) \vee Q(\ ?x,\ g(?x))$

$\neg\ P\ (?x) \vee \neg\ P\ (g(?x))$

(9) Rename variables: We want to have distinct variables in different clauses.

$\neg\ P\ (?x1) \vee \neg\ P(\ ?y) \vee P[f(?x,\ ?y)]]$

$\neg\ P\ (?x2) \vee Q(\ ?x2,\ g(?x2))$

$\neg\ P\ (?x3) \vee \neg\ P\ (g(?x3))$

Ex: Whoever can read is literate,

Dolphins are not literate.

Some dolphins are intelligent.

Goal: Some who are intelligent can not read.

Approach:

1. Form an ontology, convert problem to symbolic description.

2. Negate the goal.

3. Convert facts and negated goal into clauses.

4. Add clauses to a list.

5. Perform resolution until NIL.

Ontology:

R(?x) $\rightarrow$?x can read

L(?x) $\rightarrow$?x is literate

D(?x) $\rightarrow$?x is a dolphin

I(?x) $\rightarrow$?x is intelligent.

$(\forall ?x)\ [\ R(?x) \Rightarrow L(?x)\ ]$

$(\forall ?x)\ [\ D(?x) \Rightarrow \neg\ L(?x)\ ]$

$(\exists ?z)\ \ [D(?z) \wedge I(?z)]$

} facts

$(\exists ?x)\ \ [I(?x) \wedge \neg\ R(?x)\ ]$ Goal

Negate Goal:

$\neg\ (\exists ?x)\ \ [I(?x) \wedge \neg\ R(?x)\ ]$

make facts and negated goal into clauses:

$(\forall ?x)\ [\neg\ R(?x) \vee L(?x)\ ]$

F1: $[\neg\ R(?x) \vee L(?x)\ ]$

F2: $[\neg\ D(?y) \vee \neg\ L(?y)\ ]$

F3: $D(A) \wedge I(A)$

F3a: $D(A)$

F3b: $I(A)$

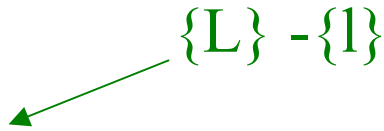
Goal: $(\forall ?x)\ [\neg\ I(?x) \vee R(?x)\ ]$

$\neg\ I(?x) \vee R(?x)$

$\neg\ I(?z) \vee R(?z)$

Apply Resolution until NIL:

F2+ F3a S={A/?y}



resolvent 5: { ¬ L(?y)} S ∪ ϕ

¬ L(A)

F1+5: S={A/?x}

resolvent 6: {¬ R(?x)}S

¬ R(A)

Goal clause + 6: S={A/?z}

resolvent 7: ¬ I(A)

F3b + 7: S={ }

resolvent 8: NIL → done.

Resolution Algorithm:

1. Negate the goal.

2. Convert goal and facts into clauses.

3. Create a list called CLAUSES, containing all clauses from step 2.

4. UNTIL NIL is a member in CLAUSES

5. BEGIN

6. Select two distinct resolvable clauses

   a_i and c_j in CLAUSES.

If there aren't any resolvable clauses return failure.

7. $r_{ij} \leftarrow$ resolvent of a_i and c_j

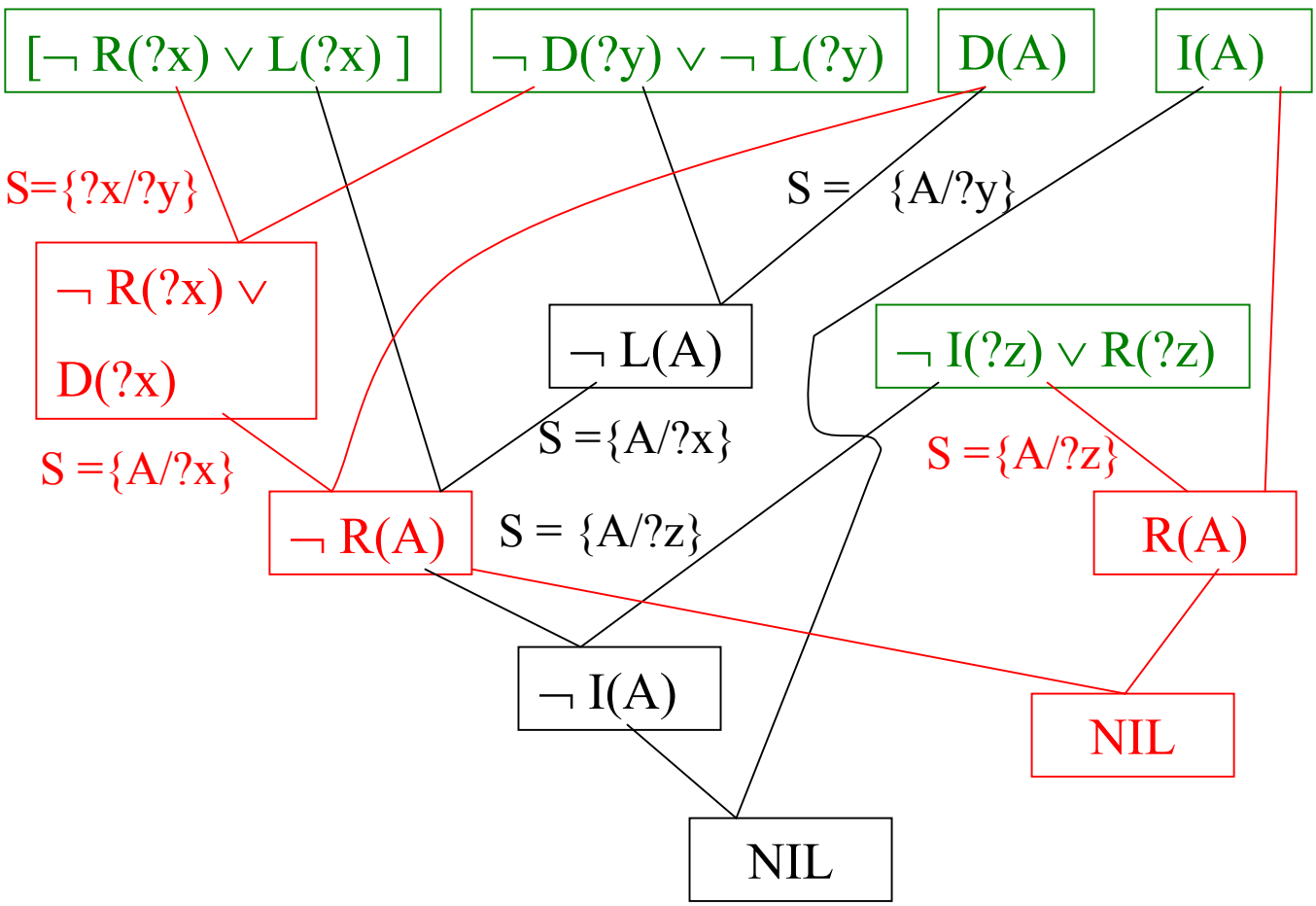
8. Add r_{ij} to the list of clauses.

9. END

We implement resolution using a derivation graph.

Derivation graph: has nodes which are each one clause, and edges link two parent clauses to a child clause which is their resolvent.

A solution is a subgraph of derivation graph which ends with NIL and it is called a refutation tree.



Ex:

(1) Fido is at whenever John is.

(2) John is at school.

(3) Where is fido? Goal

Ontology:

AT (?x, ?y) → ?x is at place ?y.

Model in FOPL:

$(\forall ?x)\ [AT\ (John, ?x) \Rightarrow AT\ (Fido, ?x)]$ (1)

AT (John, School) (2)

$(\exists ?x)\ [\ AT(\ Fido, ?x)]$ Goal

negate goal:

$\neg\ (\exists ?x)\ [\ AT(\ Fido, ?x)]$

Convert to Clauses:

(1) $(\forall ?x)\ [\neg\ AT\ (John, ?x) \vee AT\ (Fido, ?x)]$

$[\neg\ AT\ (John, ?x) \vee AT\ (Fido, ?x)]$

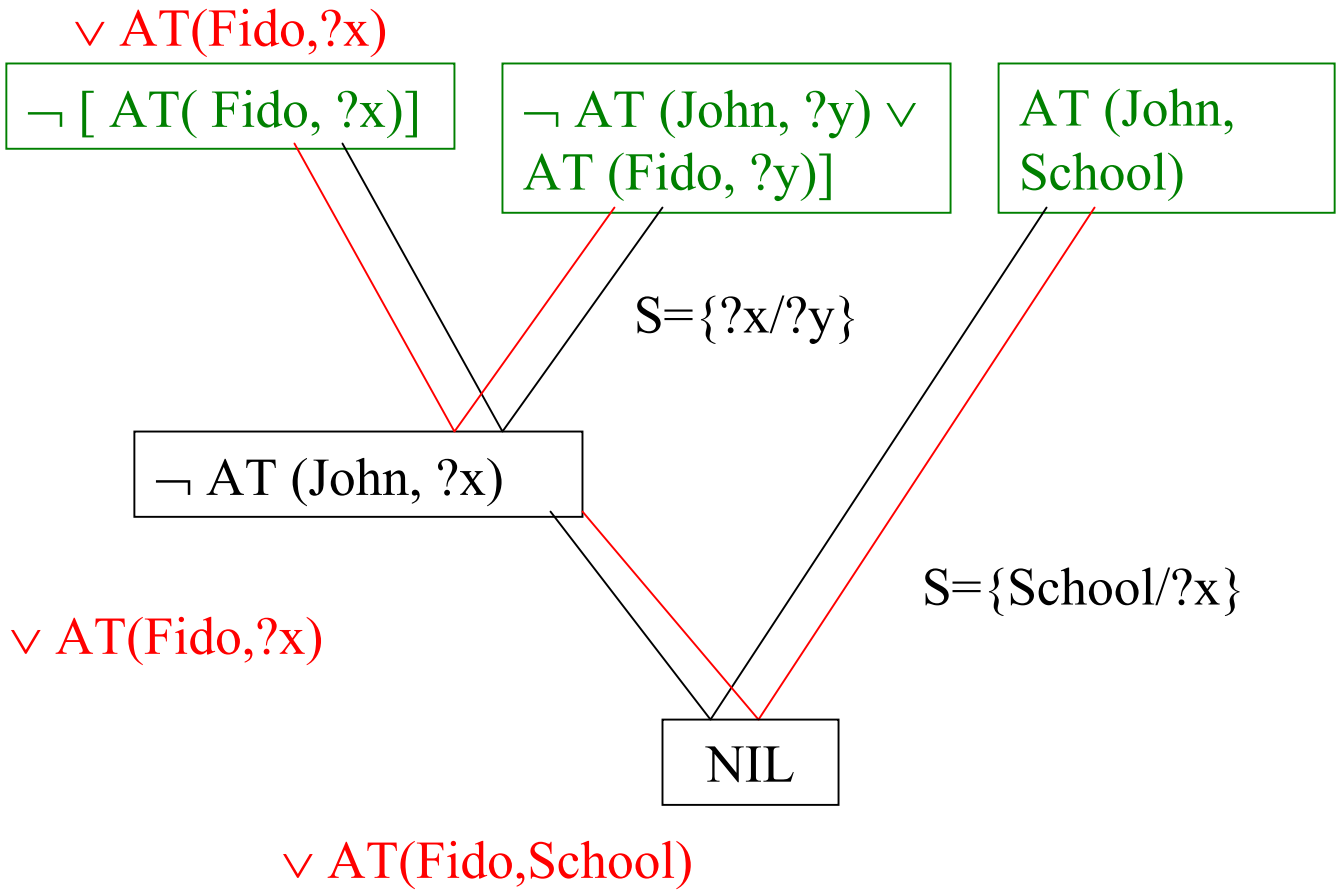
$[\neg\ AT\ (John, ?y) \vee AT\ (Fido, ?y)]$

(2) AT (John, School)

Goal: ¬ (∃?x) [AT(Fido, ?x)]

↓ (∀?x) ¬ [AT(Fido, ?x)]

↓ ¬ [AT(Fido, ?x)]



To extract Answers from Resolution Proofs:

(1) Append to each clause arising from the negation of the goal its own negation.

(this makes it a tautology).

(2) follow the structure of the refutation tree, and perform the same resolutions.

(3) The remaining clause at the root of tree will be the answer.

Strategies for control of resolution:

1. Breadth first strategy

Layer by Layer enumerate everything in the derivation graph.

2. Set of support strategy

Always take one of parent to be a goal clause or derived from a goal clause.

3. Linear input strategy

Always take one parent to be from the initial bare set of clauses.

4. Unit preference strategy.

Like the end of support strategy, but select single_literal clauses to be a parent if possible.

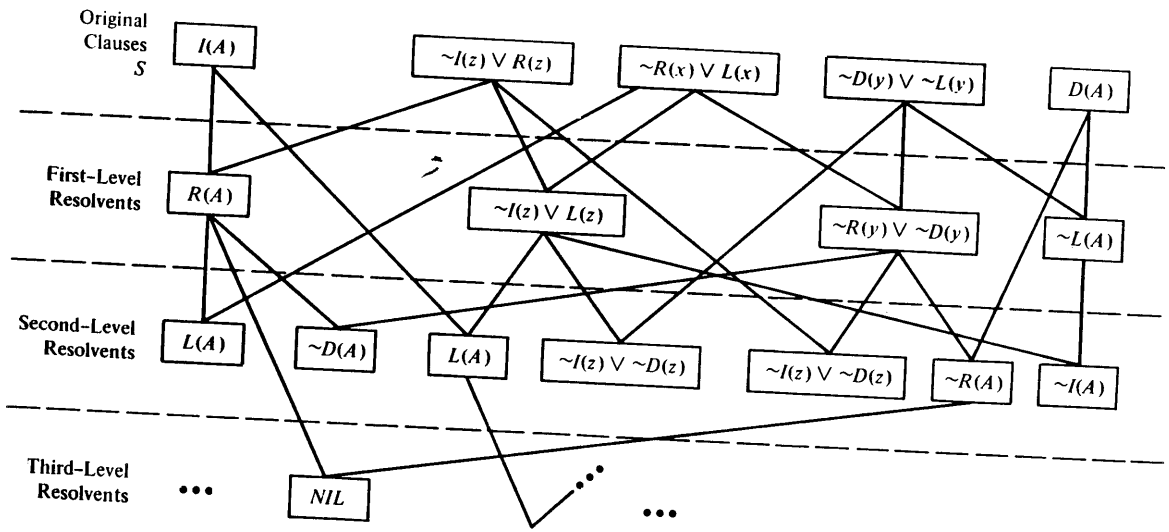


Illustration of a breadth-first strategy.

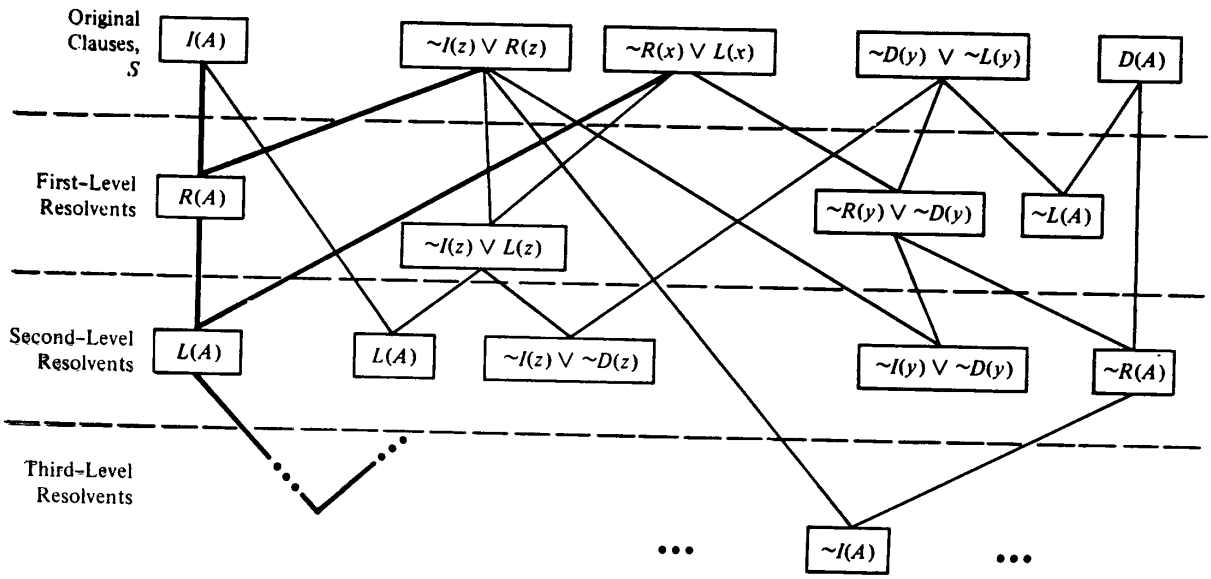


Illustration of a linear-input form strategy.

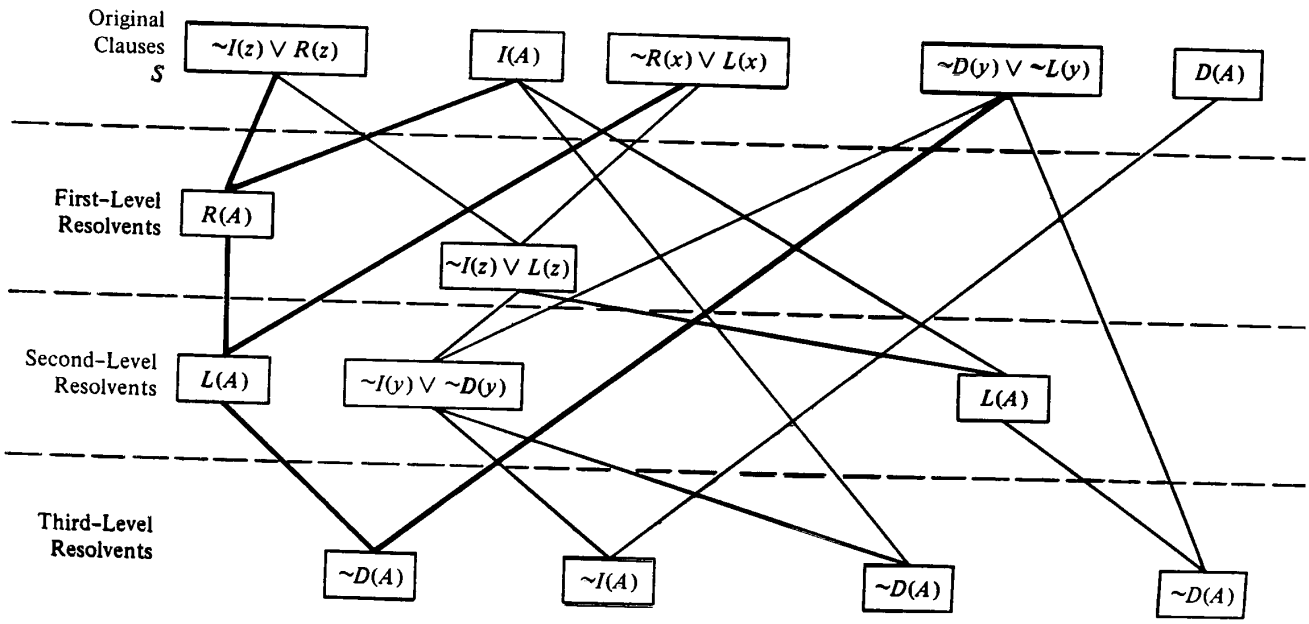


Illustration of a set-of-support strategy.

Homework

#1(a)

Goal: $(\exists ?x)\{[P(?x) \Rightarrow P(A)] \wedge [P(?x) \Rightarrow P(B)]\}$

negate goal:

$\neg (\exists ?x)\{[P(?x) \Rightarrow P(A)] \wedge [P(?x) \Rightarrow P(B)]\}$

$(\forall ?x) \neg\{[P(?x) \Rightarrow P(A)] \wedge [P(?x) \Rightarrow P(B)]\}$

$(\forall ?x) \neg\{[\neg P(?x) \vee P(A)] \wedge [\neg P(?x) \vee P(B)]\}$

$(\forall ?x) \{[P(?x) \wedge \neg P(A)] \vee [P(?x) \wedge \neg P(B)]\}$

$(\forall ?x) \{[P(?x) \wedge \neg P(A) \vee P(?x)] \wedge [P(?x) \wedge \neg P(A)] \vee \neg P(B)]\}$

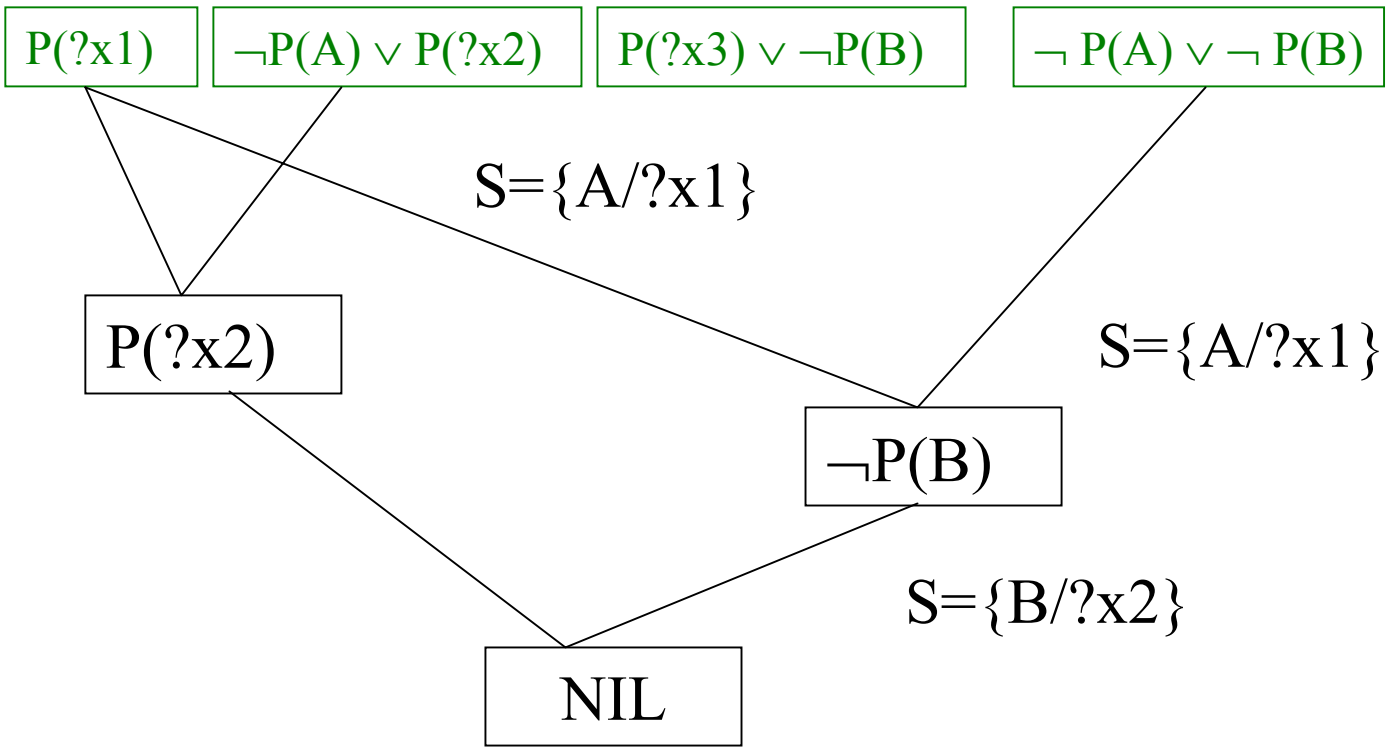
~~$(\forall ?x)$~~ $\{[P(?x) \vee P(?x)] \wedge [\neg P(A) \vee P(?x)] \wedge [P(?x) \wedge \neg P(B)] \wedge [\neg P(A) \vee \neg P(B)]\}$

C1: $P(?x1)$

C2: $\neg P(A) \vee P(?x2)$

C3: $P(?x3) \vee \neg P(B)$

C4: $\neg P(A) \vee \neg P(B)$


P(?x1)
¬P(A) ∨ P(?x2)
P(?x3) ∨ ¬P(B)
¬ P(A) ∨ ¬ P(B)
S={A/?x1}
P(?x2)
S={A/?x1}
¬P(B)
S={B/?x2}
NIL

#4 Ontology:

InAC(?x) → ?x belongs to Alpine club.

MC(?x) → ?x is a mountain climber.

Skier(?x) → ?x is a skier.

Like(?x, ?y) → ?x likes ?y.

goal: (∃?x)[InAC(?x) ∧MC(?x) ∧ ¬Skier(?x)]

facts:

InAC(Tony)

InAC(Mike)

InAC(John)

Like(Tony, Rain)

Like(Tony, Snow)

Rules: W⇒L

R1: (∀?x2){[InAC(?x2) ∧ ¬Skier(?x2)] ⇒ MC(?x2)}

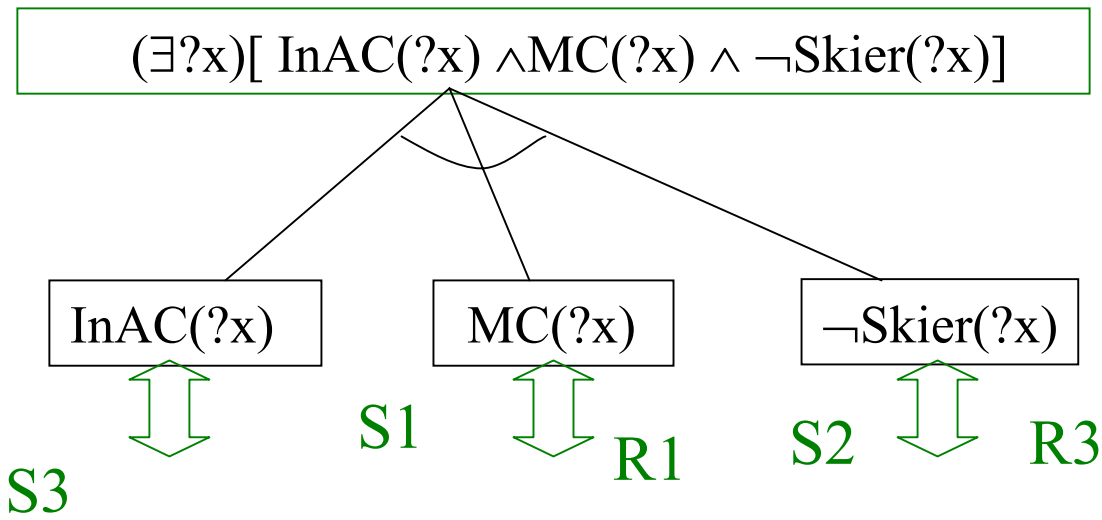
R2: (∀?x3)[MC(?x2)}⇒ ¬Like(?x3, Rain)]

R3: (∀?x4)[¬Like(?x4,Snow) ⇒ ¬Skier(?x4)]

R4: (∀?x5)[Like(Tony,?x5) ⇒ ¬Like(Mike,?x5)]

R5: (∀?x6)[¬ Like(Tony,?x6) ⇒ Like(Mike,?x6)]

AND-OR Graph:



Can match with a fact.

Ex: family tree.

Everyone has a parent.

For all ?x and ?y; if ?x is parent of ?y, and ?y is parent of ?z, then ?x is grandparent of ?z.

Can you find two individuals ?u, and ?v, such that ?u is grandparent of ?v.

Ontology:

$P(?x, ?y) \rightarrow$?x is parent of ?y

$G(?x, ?y) \rightarrow$?x is grandparent of ?y.

Facts:

$(\forall ?y)\ (\exists ?x)\ [\ P(?x, ?y)]$

$(\forall ?x)\ (\forall ?y)\ (\forall ?z)\ [\ P(?x, ?y) \wedge P(?y, ?z)]$

$\Rightarrow G(?x, ?z)]$

Goal:

$(\exists ?u)\ (\exists ?v)\ [\ G(?u,?v)]$

negate Goal:

$\neg\ (\exists ?u)\ (\exists ?v)\ [\ G(?u,?v)]$

$(\forall ?u)(\forall ?v)[\neg\ G(?u,?v)]$

$\neg\ G(?u,?v)$

Clause form for facts:

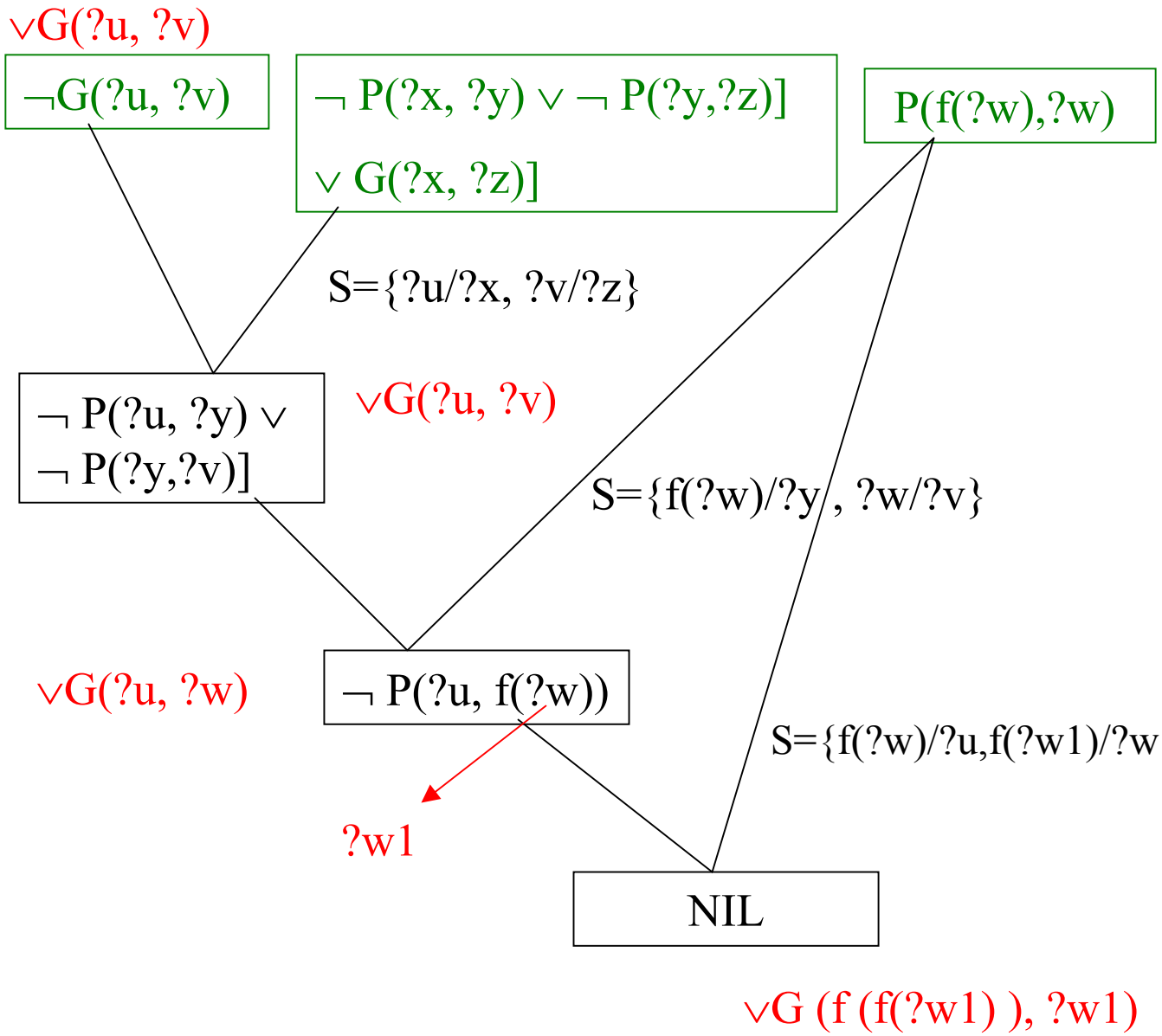
f1: $(\forall ?y)\ \ P(\ f\ (?y),\ ?y)$

$P(\ f(?y),\ ?y)$ —Rename ?y to ?w→ $P(\ f(?w),\ ?w)$

f2: $(\forall ?x)\ (\forall ?y)\ (\forall ?z)\ [\neg\ P(?x,\ ?y) \vee \neg\ P(?y,\ ?z)]$

$\vee\ G(?x,\ ?z)]$

$\neg\ P(?x,\ ?y) \vee \neg\ P(?y,\ ?z)] \vee\ G(?x,\ ?z)]$


∨G(?u, ?v)
¬G(?u, ?v)
¬ P(?x, ?y) ∨ ¬ P(?y,?z)]
∨ G(?x, ?z)]
P(f(?w),?w)
S={?u/?x, ?v/?z}
¬ P(?u, ?y) ∨
¬ P(?y,?v)]
∨G(?u, ?v)
S={f(?w)/?y, ?w/?v}
∨G(?u, ?w)
¬ P(?u, f(?w))
S={f(?w)/?u,f(?w1)/?w}
?w1
NIL
∨G (f (f(?w1)), ?w1)

If Goal is:

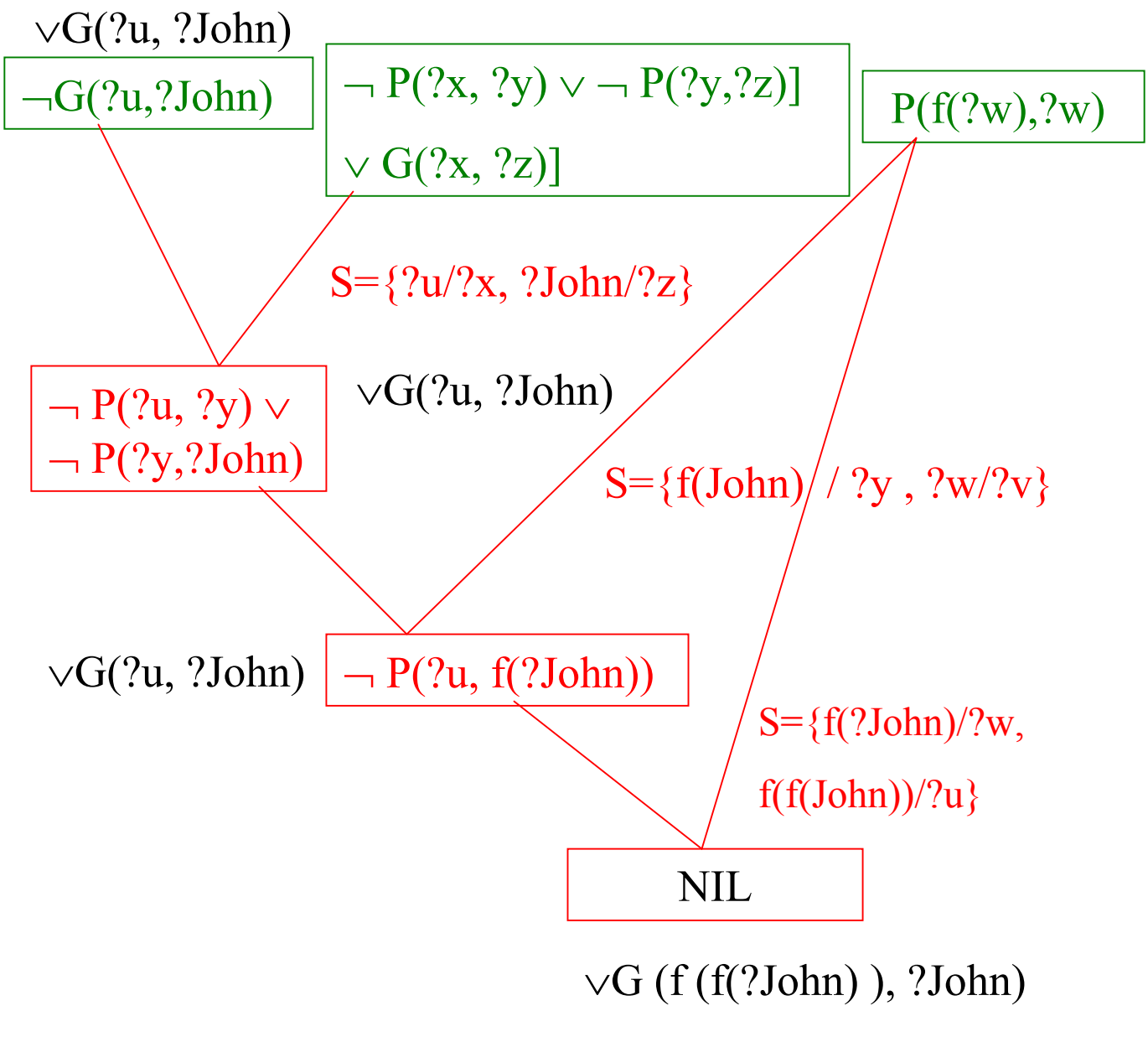
$(\exists ?u)\ [\ G(?u,John)]$

then:

$\neg\ (\exists ?u)\ [\ G(?u,John)]$

$\downarrow$

$\neg\ G(?u,John)$



Automatic Problem Solving- Planning

We have a database of predicates describing the world symbolically.

In each iteration, we apply actions. Actions both add and retract expressions from the world descriptions. We discuss automatic problem solving using examples from robots.

Task is:

1) Given knowledge of the world,

2) formulate a plan of actions.

3) Monitor the execution for correctness.

Architecturally we have:

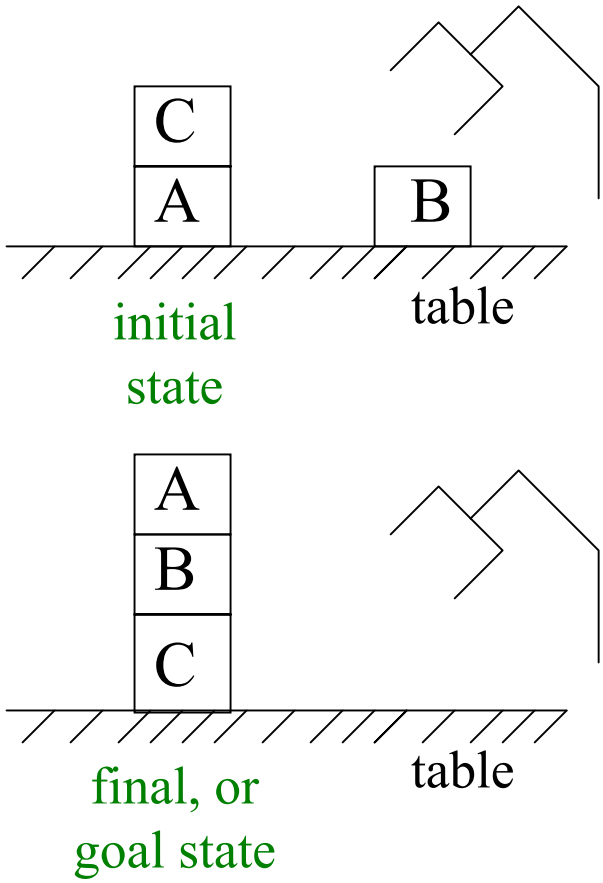
1. Database - State of world described in FOPL.

2. Actions - Described by lists of predicates.

3. Search mechanism - Decides what actions to apply when there is a tie.

<<< Refer to Chapter 11 of the text >>>

Ex: Robots:



Ontology:

Clear(?x) → top of block ?x is clear.

Handempty → no object is in robot's gripper

On (?x, ?y) → Block ?x is on block ?y.

Ontable (?x) → ?x is on the table.

Apply Ontology to describe the example.

Initial State:

Clear(B) ∧ Handempty ∧ On(C,A) ∧ Clear (C) ∧ Ontable(A) ∧ Ontable (B)

Goal State:

On (B,C) ∧

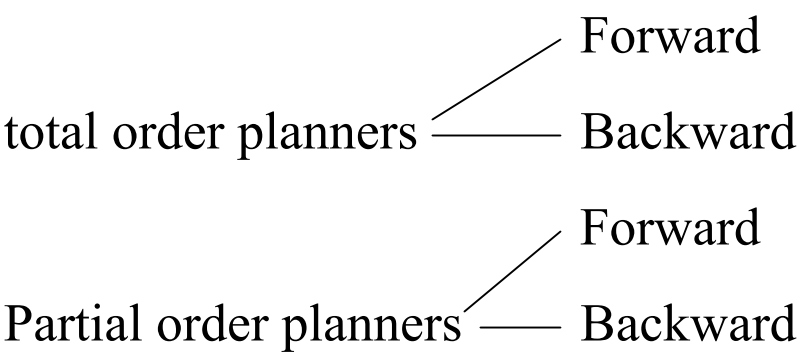
On (A,B) ∧

Clear (A) ∧

Ontable (C) ∧

Handempty

Planners can be classified into different categories:



Hierarchical planners

Non-hierarchical

Assumptions for total order planners discussed here:

1. Goals must be conjunctions of predicates.

$$G = L1 \wedge L2 \wedge \ldots\ldots\ldots \wedge Ln.$$

2. Initial and intermediate states are conjunctions of ground predicates.

3. Variables are assumed to be existentially quantified.

Solutions or Plans

Solutions or plans are a sequence of actions and their instantiations. The correct solution is the one which has the correct set of actions, the correct order between the actions, and the correct instantiation for the actions.

Actions:

Things like Pickup (?x)

Solutions:

Actions, their instantiations, order between actions.

each action is represented by three lists (STRIP style representation)

(i) Preconditions:

Things which must be true before we can apply the action.

(ii) Delete list:

List of predicates which are no longer true after the action is executed.

(iii) Add list :

List of predicates which are added to the database after the action is executed.

In some implementations for planning systems the Precondition list and the delete list are merged and only one list is used.

Examples

the lists modeling Pickup(?x) action:

Pickup(?x) corresponds to picking up object ?x from the table.

Precondition:

Ontable(?x) $\wedge$ Handempty $\wedge$ Clear (?x)

Delete List:

Ontable(?x) $\wedge$ Handempty $\wedge$ Clear(?x):

Add List:

Holding(?x)

A model for the action putdown(?x)

Preconditions:

Holding (?x)

Delete List:

Holding (?x)

Add List:

Clear(?x) $\wedge$ Handempty $\wedge$ Ontable (?x)

Two more actions for the example robot problem:

Stack(?x, ?y)

Preconditions: Holding(?x) ∧ Clear (?y)

Delete List: Holding (?x) ∧ Clear (?y)

Add List: On(?x, ?y) ∧ Handempty ∧ Clear (?x)

Unstack(?x, ?y)

Preconditions: Handempty ∧ On(?x, ?y) ∧ Clear (?x)

Delete List: Handempty ∧ On (?x, ?y) ∧ Clear(?x)

Add List: Clear(?x) ∧ Holding(?x) ∧ Clear(?y)

Solution for this problem:

Unstack(C,A)

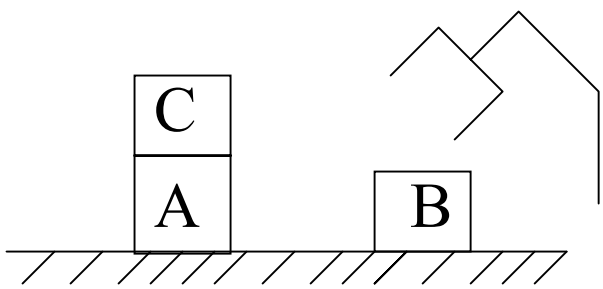
Putdown(C)

Pickup(B)

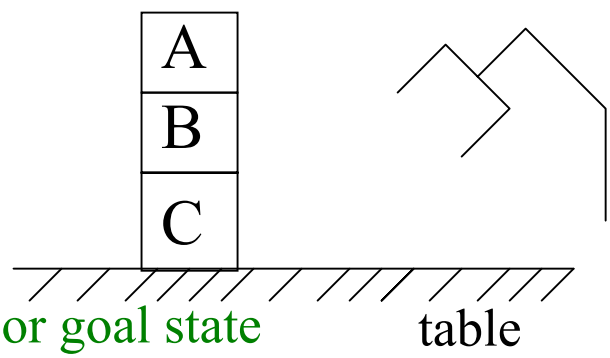
Stack(B,C)

Pickup(A)

Stack(A,B)



initial state



final, or goal state

To generate solutions/plans we do either forward planning or backward planning.

Forward Planning:

Start with the initial state and apply actions one at a time, modify the state description, until the goal state is reached.