Knowledge Representation and Reasoning

First Order Logic

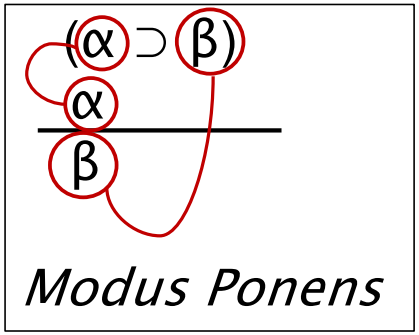
Deepak Khemani
Plaksha University

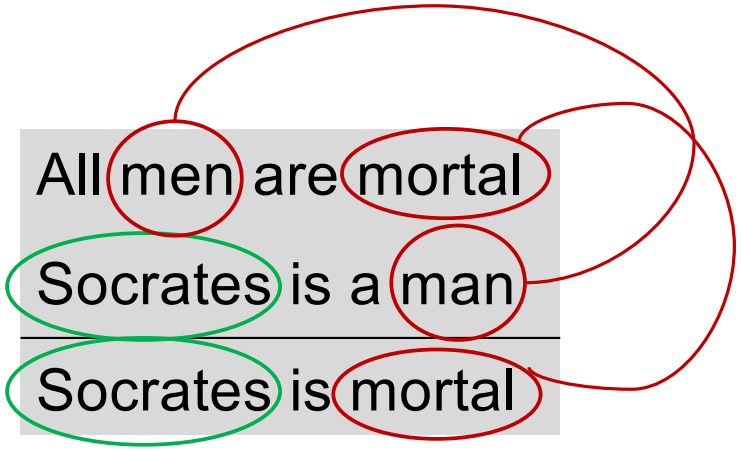
Sentences, Denotation, and Truth

- A logic is a formal language that defines a set of sentences $\{\alpha, \beta, \gamma \ldots\}$
- Each sentence has a meaning, ascribed from outside.
- Each sentence has a truth value, ascribed from outside
- Compound sentences have truth values that are determined completely by its constituents and logical operators
- Logic has a notion of Entailment
- Logic also has Rules of Inference
- Logic has a machinery or algorithms for producing new sentences from old by applying the rules of inference
- The goal of building the machinery is to mechanically produce all and only the true sentences (conclusions) – given a set of true sentences (axioms, premises)

Deconstructing Sentences

Deduction is possible because of *connections* between *constituents* of sentences





The form of the argument captures the connections

Propositional Logic cannot make these connections

Aristotelian Logic: Categories and Individuals

Aristotle was concerned with making statements about categories

- *Universal Affirmation*:	All S are P	All Men are Mortal
- *Particular Affirmation*:	Some S are P	Some Children are Tall
- *Universal Denial*:	No S is P	No Priest is Immortal
- *Particular Denial*:	Some S are not P	Some Birds are not Priests

Individuals could belong to categories. For example, "X is a P"

The Socratic Argument is *one of* the fourteen valid forms of the Syllogism

All Men are Mortal
Socrates is a Man
Therefore, Socrates is Mortal

See https://plato.stanford.edu/entries/aristotle-logic/

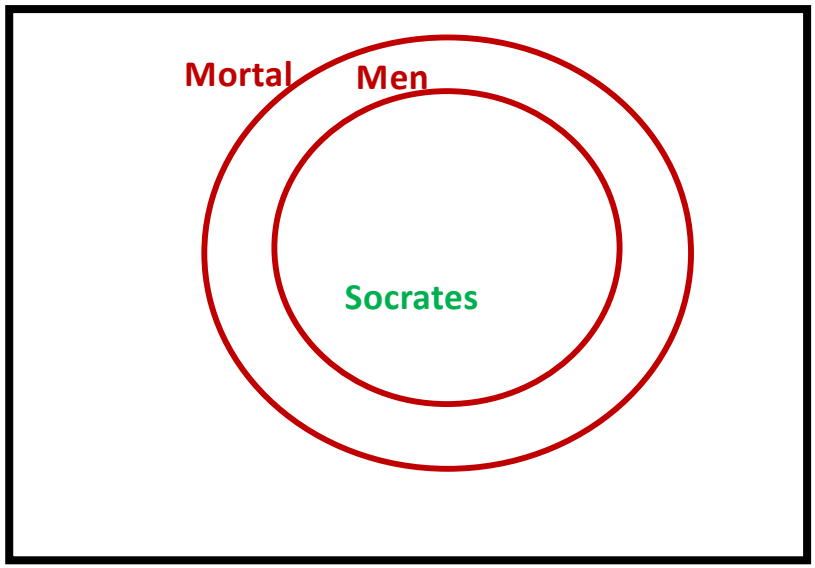
Relations on a Domain

A Domain D or a Universe of Discourse is the ”world” of First Order Logic

All men are mortal

Socrates is a man

Socrates is mortal



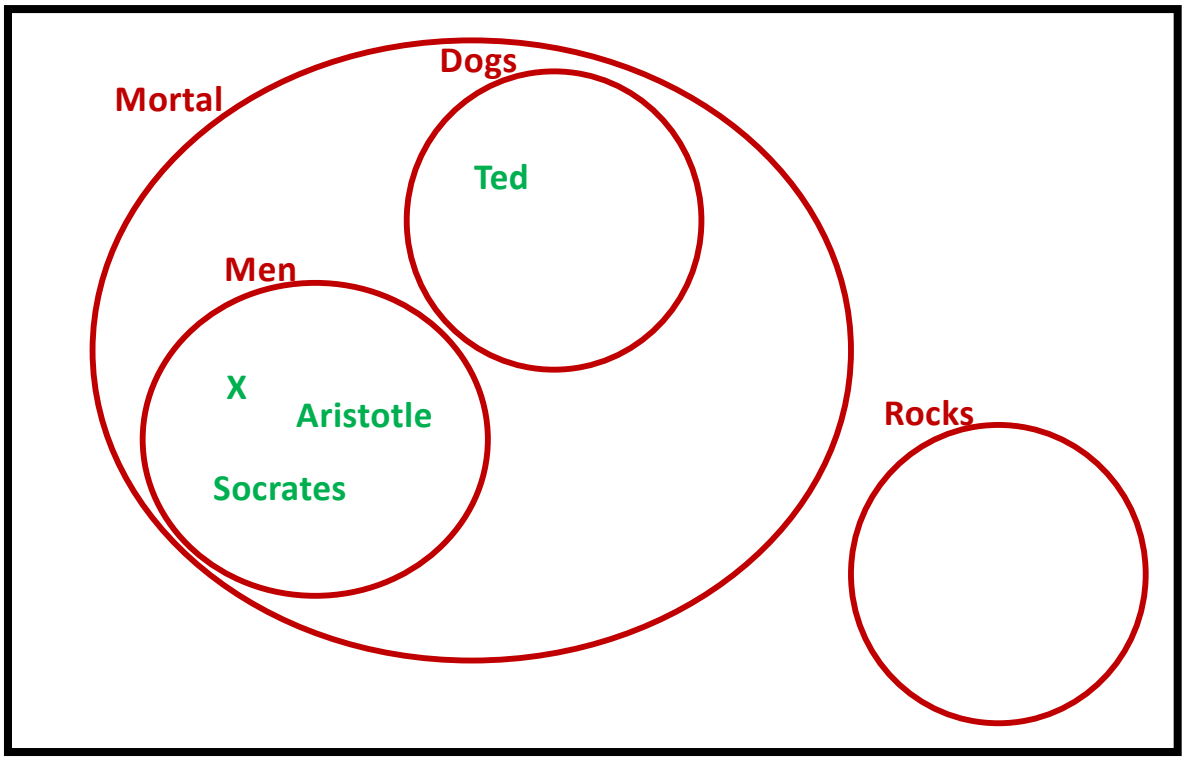
Categories are unary relations or subsets of D

First Order Logic / Predicate Calculus

- The meaning or semantics in PL was externally defined.
- The semantics of First Order Logic (FOL) is defined over a domain.
- The domain D is a set and has individuals
  - Individuals may be named and identified
  - Individuals maybe unknown or variables
- FOL is also called Predicate Calculus because of the use of predicates.
  - Unary predicates are interpreted as subsets of the domain D
  - Binary predicates are binary relations. Subsets of DxD.
- FOL is a logic that breaks down a sentence into constituents and relations between them.
- FOL also has the vocabulary to talk of “all” and “some”

Predicates and Individuals

Individuals
Constants (named) – Aristotle
Variables (unknown) – X



Predicates
Unary relations
Man(Socrates)
Man(Aristotle)
Man(X)
Dog(Ted)
Binary relations
Teacher(Socrates, Aristotle)
Pet(Socrates,Ted)

First Order Logic (FOL): Syntax

The *logical* part of the vocabulary

- Symbols that stand for connectives or operators
  - "∧", "∨", "¬", and "⊃"…
- Brackets "(", ")", "{", "}"…
- The constant symbols "⊥" and "⊤".

- A set of variable symbols $V = \{v_1, v_2, v_3, \ldots\}$
  - commonly used $\{x, y, z, x_1, y_1, z_1, \ldots\}$
- Quantifiers: "∀" read as "for all", and "∃" read as "there exists".
  The former is the *universal quantifier* and
  the latter the *existential quantifier*.
- The symbol "=" read as "equals".

FOL Syntax (contd)

The non-logical part of *FOL vocabulary* constitutes of three sets.

- A set of predicate symbols P = {P_1, P_2, P_3, …}. We also use the symbols {*P*, *Q*, R, …}. More commonly we use words like “Man”, “Mortal”, “GreaterThan”. Each symbol has an arity associated with it.
- A set of function symbols F = {f_1, f_2, f_3, …}. We commonly used the symbols {f, g, h…} or words like “Successor” and “Sum”. Each function symbol has an arity that denotes the number of argument it takes.
- A set of constant symbols C = {c_1, c_2, c_3, …}. We often used symbols like “0”, or “Socrates”, or “Darjeeling” that are meaningful to us.

The three sets define a language L(*P*,*F*,*C*) or L(*R*,*F*,*C*)

Terms of L(*P*,*F*,*C*)

The basic constituents of *FOL* expressions are *terms*. The set of terms $\mathfrak{I}$ of L(*P*,*F*,*C*) is defined as follows. The constants and the variables are terms by definition. More terms are defined using the function symbols.

If $t \in V$ then $t \in \mathfrak{I}$

If $t \in C$ then $t \in \mathfrak{I}$

If $t_1, t_2, \ldots, t_n \in \mathfrak{I}$ and $f \in F$ is an n-place function symbol
then $f(t_1,t_2, \ldots,t_n) \in \mathfrak{I}$

Atomic Formulas of L(P,F,C)

The set of formulas is defined using terms and predicate symbols. By default the logical symbols "$\bot$" and "$\top$" are also formulas. The set of well formed formulas *F* of L(*P*,*F*,*C*) is defined as follows.

Atomic formulas $\mathcal{A}$

$\bot \in \mathcal{A}$

$\top \in \mathcal{A}$

If $t_1, t_2 \in \Im$ then $(t_1=t_2) \in \mathcal{A}$

If $t_1, t_2, \ldots, t_n \in \Im$
and $P \in$ P is an n-place predicate symbol
then $P(t_1,t_2,\ldots,t_n) \in \mathcal{A}$

Formulas of L(P,F,C)

The set of formulas of L(P,F,C) $\mathcal{F}$ is defined as follows

If $\alpha \in \mathcal{A}$ then $\alpha \in \mathcal{F}$

If $\alpha \in \mathcal{F}$ then $\neg\alpha \in \mathcal{F}$

If $\alpha, \beta \in \mathcal{F}$ then $(\alpha \wedge \beta) \in \mathcal{F}$

If $\alpha, \beta \in \mathcal{F}$ then $(\alpha \vee \beta) \in \mathcal{F}$

If $\alpha, \beta \in \mathcal{F}$ then $(\alpha \supset \beta) \in \mathcal{F}$

Universal and Existential Quantifiers

If $\alpha \in \mathcal{F}$ and $x \in V$ then $\forall x\ (\alpha) \in \mathcal{F}$

$\forall x\ (\alpha)$ is read as "for all x (α)"

If $\alpha \in \mathcal{F}$ and $x \in V$ then $\exists x\ (\alpha) \in \mathcal{F}$

$\exists x\ (\alpha)$ is read as "there exists x (α)"

We will also use the notation (forall (x) (α)) and (exists (x) (α)) as given in the book Artificial Intelligence by Eugene Charniak and Drew McDermott.

Makes representation for use in programs simpler.

List notation

Standard mathematical notation

1. $\forall x$ (Man(x) $\supset$ Human(x)) : all men are human beings
2. Happy(suresh) $\vee$ Rich(suresh) : Suresh is rich or happy
3. $\forall x$ (CitrusFruit(x) $\supset \neg$Human(x)) : all citrus fruits are non-human
4. $\exists x$ (Man(x) $\wedge$ Bright(x)) : some men are bright

List notation (a la Charniak & McDermott, "Artificial Intelligence")

1.(forall (x) (if (man x) (human x)))

2.(or (happy suresh) (rich suresh))

3.(forall (x) (if (citrusFruit x) (not (human x))))

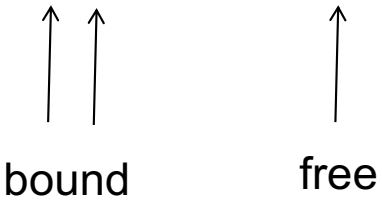
4.(exists (x) (and (man x) (bright x)))

Sentences of L(P,F,C)

A variable within the scope of a quantifier is said to be *bound*.

If a variable is not bound then it is *free*.

Example: (forall (x) (and (exists (y) (loves-to-read x y)) (book y))



A formula of L(P,F,C) without free variables is a sentence of L(P,F,C)

FOL: Rules of Inference

The propositional logic rules we saw earlier are valid in *FOL* as well. In addition we need new rules to handle quantified statements. The two commonly used rules of inference are,

$$\frac{\forall x\, P(x)}{P(a)} \quad \text{where } a \in C \quad \text{Universal Instantiation (UI)}$$

$$\frac{P(a)}{\exists x\, P(x)} \quad \text{where } a \in C \quad \text{Generalization}$$

Examples:

$$\frac{\forall x\, (Man(x) \supset Mortal(x))}{(Man(Socrates) \supset Mortal(Socrates))}$$

$$\frac{(Man(Socrates) \supset Mortal(Socrates))}{\exists x\, (Man(x) \supset Mortal(x))}$$

FOL: Rules of Substitution

The following rules of substitution are also useful,

$\neg\forall x\ \alpha$	$\equiv$	$\exists x\ \neg\alpha$	DeMorgan's law
$\neg\exists x\ \alpha$	$\equiv$	$\forall x\ \neg\alpha$	DeMorgan's law
$\forall x\ \forall y\ \alpha$	$\equiv$	$\forall y\ \forall x\ \alpha$	
$\exists x\ \exists y\ \alpha$	$\equiv$	$\exists y\ \exists x\ \alpha$	

Semantics (First Order Logic)

Difficult to express universal statements meaningfully in Propositional Logic. Consider,

Alice likes mathematics and she likes stories. If she likes mathematics she likes algebra. If she likes algebra and likes physics she will go to college. …

The statements in red colour are specific to Alice. We often want to make these as general statements - *If SOMEONE likes mathematics she likes algebra. If SOMEONE likes algebra and likes physics she will go to college.*

To make such general statements and reason with them we need the notion of variables that FOL gives us, and the universal and existential quantifiers.

The variables take values from a domain, and thus we have the notion of Interpretations in FOL where we choose a domain and interpret the language L(P,F,C) over the domain.

Semantics: Interpretations for L(P,F,C)

An Interpretation ϑ = <D, I> of a FOL language L(P,F,C) constitutes of a domain (or Universe of Discourse) *D* and a mapping *I* from the language *L* to the domain *D*.

Each of the elements of the sets *P, F* and *C* are interpreted over *D*. Each of them is understood or gets meaning from the domain *D*.

Predicate symbols are mapped to relations on *D*

Function symbols are mapped to functions on *D*

Constant symbols are mapped to individuals in *D*

Interpretation ϑ = <D,I> of L(P,F,C)

For every predicate symbol $Q \in P$ of arity N,

$I(Q) = Q^I$ where Q^I is the image of Q and $Q^I \subseteq DxDx\ldots xD$

For every function symbol $f \in F$ of arity N,

$I(f) = f^I$ where f^I is the image of f and f^I: $DxDx\ldots xD \rightarrow D$

For every constant symbol $c \in C$

$I(c) = c^I$ where c^I is the image of c and $c^I \in D$

In addition we have an assignment A: $V \rightarrow D$

from the set of variables of L(P,F,C) to the domain.

$A(v) = v^A$ where $v^A \in D$

Interpretation of Terms of L(P,F,C)

Terms in FOL *denote* elements in the domain.

A term $t \in \Im$ mapped to the element of the domain *D* as follows.

If $t \in V$ then $t^{IA} = t^{A}$

If $t \in C$ then $t^{IA} = t^{I}$

If $t = f(t_1,t_2, \ldots,t_n)$ and $f \in F$ then $t^{IA} = f^{I}(t_1^{IA}, t_2^{IA}, \ldots , t_n^{IA})$

Variables are mapped by the assignment A.

For example, x → 12

Constants are interpreted by the mapping I.

For example, sifar → 0

Functions denote elements too.

For example, plus(3,8) → 11

Interpretations

Consider the following KB

{(O A B), (O B C), (not (M A)), (M C)}

or {O(A, B), O(B, C), ¬M(A), ¬M(C)}

Where O is a binary predicate symbol
M is a unary predicate symbol
A, B and C are constant symbols

What is the above KB describing?

Observe that we have not used helpful names!

Remember – meaning lies in the mind of the beholder

Interpretation 1

{(O A B), (O B C), (not (M A)), (M C)}

Domain: Blocks World

Predicate symbols

O → On

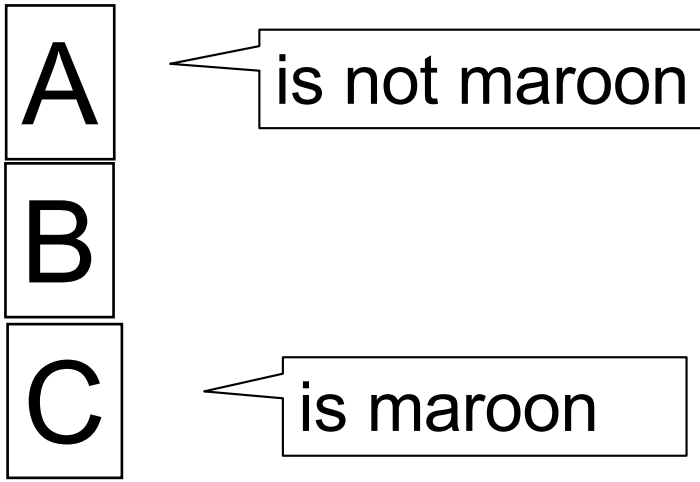
M → Maroon

Constant Symbols

A, B, C → blocks

A is on B

B is on C



Interpretation 2

{(O A B), (O B C), (not (M A)), (M C)}

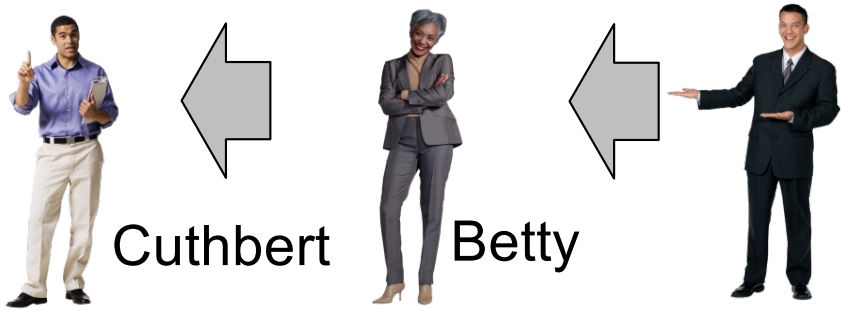
Domain: People

Predicate symbols
O → LookingAt
M → Married

Constant Symbols
A → Archie
B → Betty
C → Cuthbert

Betty is looking at Cuthbert

Archie is looking at Betty



is married

is not married

Truth Assignment to Atomic Formulas of FOL

A valuation function Val: $\mathcal{F} \rightarrow$ {*true, false*}

$Val(\top) = true$

$Val(\bot) = false$

$Val(t_1 = t_2)^{IA} = true$ iff $t_1^{IA} = t_2^{IA}$

$Val(Q(t_1, t_2, \ldots, t_n))^{IA} = \text{true iff } \langle t_1^{IA}, t_2^{IA}, \ldots, t_n^{IA} \rangle \in Q^I$

For example,

- colour(lily) = white is *true* iff both refer to the same colour
- president(usa) = commander(us_army) is *true* iff both are the same person
- LessThan(5,17) is *true* iff <5,17> ∈ < relation on Natural Numbers
- Brother(suresh, ramesh) is *true* iff
  <suresh, ramesh> ∈ Brother relation on the set of people

Truth Assignment to Formulas of FOL

Logical connectives are interpreted in the standard way

If Val(α) = *true* then Val(¬α) = *false*

If Val(α) = *false* then Val(¬α) = *true*

If Val(α) = *false* and Val(β) = *false* then Val(α∨β) = *false*
else Val(α∨β) = *true*

If Val(α) = *true* and Val(β) = *false* then Val(α⊃β) = *false*
else Val(α⊃β) = *true*

If Val(α) = *true* and Val(β) = *true* then Val(α∧β) = *true*
else Val(α∧β) = *false*

Truth Assignment to Quantified Formulas of FOL

A formula of the form $\exists x(\alpha)$ is true if there is some value of *x* for which the formula is true. A universally quantified formula $\forall x\ (\alpha)$ is true for all possible values of *x*. Formally,

$(\exists x\ (\alpha))^{IA}$ = true iff α^{IB} is true for *some* assignment B that is an x-variant of A.

In other words the formula α is true for some value of *x*.

$(\forall x\ (\alpha))^{IA}$ = true iff α^{IB} is true for *all* assignments B that are x-variants of A.

In other words the formula α is true for all values of *x*.

An assignment *B* is said to be an x-variant of an assignment *A* if they agree on the value of all variables except *x*.

Truth Assignment to Sentences of FOL

- A sentence in FOL is a formula without any free variables.
- This means that all variables in the formula are quantified.
- As a consequence the sentences are *true* or *false* independent of the assignment mapping.

The meaning of the terms and sentences of a set of *FOL* sentences is given by an *interpretation* ϑ = <D, I>, where *D* is a domain and *I* is an interpretation mapping.

An interpretation M = <D, I> of set of sentences
or a theory in a language L(*P*,*F*,*C*) is a *model*
if all the sentences in the set are *true* in the interpretation.

Tautologies, Satisfiable and Unsatisfiable formulas

- A **tautology** is a formula of L(P,F,C) that is *true* in all interpretations.
  - For example Happy(suresh) $\vee \neg$ Happy(suresh)
- A formula of L(P,F,C) is **satisfiable** iff it is *true* in at least one interpretation.
  - For example $\forall$x (Man(x) $\supset$ Human(x))
    (depends on the meaning of Man and Human)
- A formula of L(P,F,C) is **unsatisfiable** iff it is *true* in no interpretation.
  - For example Happy(suresh) $\wedge \neg$ Happy(suresh)

Proofs

(in First Order Logic)

FOL: Will Alice go to college?

Let us rephrase our example (Alice) problem in first order terminology.

- Alice likes mathematics and she likes stories.
- If someone likes mathematics she likes algebra[1].
- If someone likes algebra and likes physics she will go to college.
- Alice does not like stories or she likes physics.
- Alice does not like chemistry and history."

We can formalize the statements in *FOL* as follows.

1. likes(Alice, Math) $\wedge$ likes(Alice, stories)
2. $\forall$x(likes(x, Math) $\supset$ likes(x, Algebra))
3. $\forall$x((likes(x, Algebra) $\wedge$ likes(x, Physics)) $\supset$ goesTo(x, College))
4. $\neg$likes(Alice, stories) $\vee$ likes(Alice, Physics)
5. $\neg$likes(Alice, Chemistry) $\wedge$ $\neg$likes(Alice, History)

[1] Here we must emphasize that *she* stands for both *she* and *he*.

Forward Chaining

From Facts to Goals

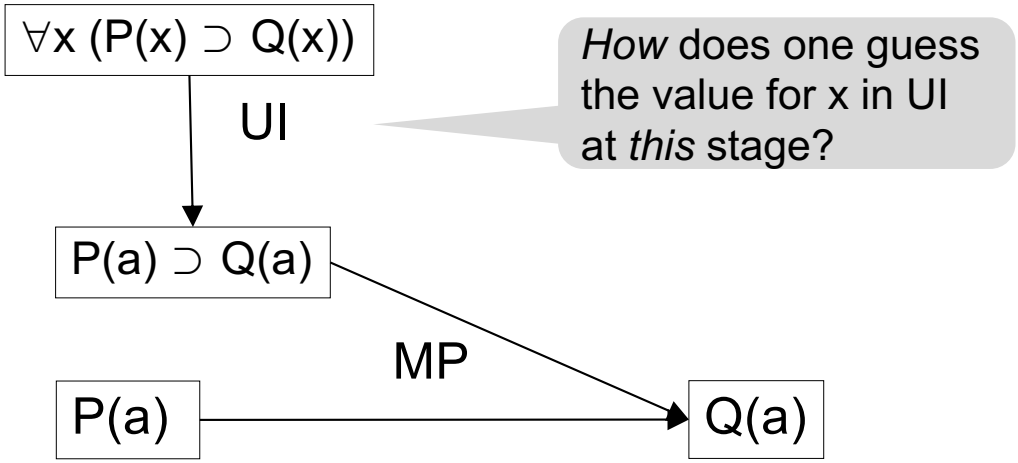
The FOL Proof Natural Deduction

1. likes(Alice, Math) ∧ likes(Alice, stories)
2. ∀x(likes(x, Math) ⊃ likes(x, Algebra))
3. ∀x((likes(x, Algebra) ∧ likes(x, Physics)) ⊃ goesTo(x, College))
4. ¬likes(Alice, stories) ∨ likes(Alice, Physics)
5. ¬likes(Alice, Chemistry) ∧ ¬likes(Alice, History)

We can now generate a proof that is analogous to the proof in propositional logic.

6. likes(Alice, Math) — 1, simplification
7. likes(Alice, stories) — 1, simplification
8. (likes(Alice, Math) ⊃ likes(Alice, Algebra)) — 2, UI
9. likes(Alice, Algebra)) — 6, 8, modus ponens
10. likes(Alice, Physics) — 4, 7, disjunctive syllogism
11. ((likes(Alice, Algebra) ∧ likes(Alice,Physics)) — 9, 10, conjunction
12. ((likes(Alice, Algebra) ∧ likes(Alice,Physics)) ⊃ goesTo(Alice,College)) — 3, UI
13. goesTo(Alice, College) — 12, 11, modus ponens

Forward Chaining in FOL



Forward chaining in FOL is a *two step* process.

The use of *Implicit Quantifier Notation*
collapses this two step inference into one step.

Implicit Quantifier notation

Prefix universally quantified variables with a "?". Replace existentially quantified variables not in the scope of a universal quantified with a *Skolem constant* (named after the mathematician Thoralf Skolem)

1. Man(?x) $\supset$ Human(?x) : all men are human beings
2. Happy(suresh) $\vee$ Rich(suresh) : Suresh is rich or happy
3. CitrusFruit(?x) $\supset \neg$Human(?x) : all citrus fruits are non-human
4. Man(sk-11) $\wedge$ Bright(sk-11)) : some men are bright

List notation

1. (if (man ?x) (human ?x))
2. (or (happy suresh) (rich suresh))
3. (if (citrusFruit ?x) (not (human ?x)))
4. (and (man sk-11) (bright sk-11))

Unifier: Substitution

A *substitution* θ is a set of <variable, value> pairs each denoting the value to be substituted for the variable.

When we apply the substitution to a formula α
the new formula is denoted by $\alpha\theta$

A *unifier* for two formulas α and β is a substitution that makes the two formulas identical. We say that α *unifies* with β.

A unifier θ unifies a set of formulas $\{\alpha_1, \alpha_2, \ldots, \alpha_N\}$ if,

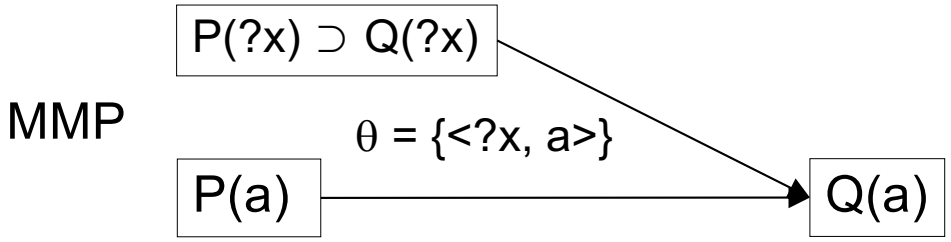
$$\alpha_1\theta = \alpha_2\theta = \ldots = \alpha_N\theta = \varphi$$

We call the common reduced form φ the *factor*.

Modified Modus Ponens (MMP)

MPP: From $(\alpha \supset \gamma)$ and β infer $\gamma\theta$ where θ is a unifier* for α and β and $\gamma\theta$ is the formula obtained by applying the substitution θ to γ.

For example,



*A substitution θ is a *unifier* for two (or more) formulas α and β if when applied it makes the two formulas identical. That is, $\alpha\theta = \beta\theta$

MPP: an example

Thus if

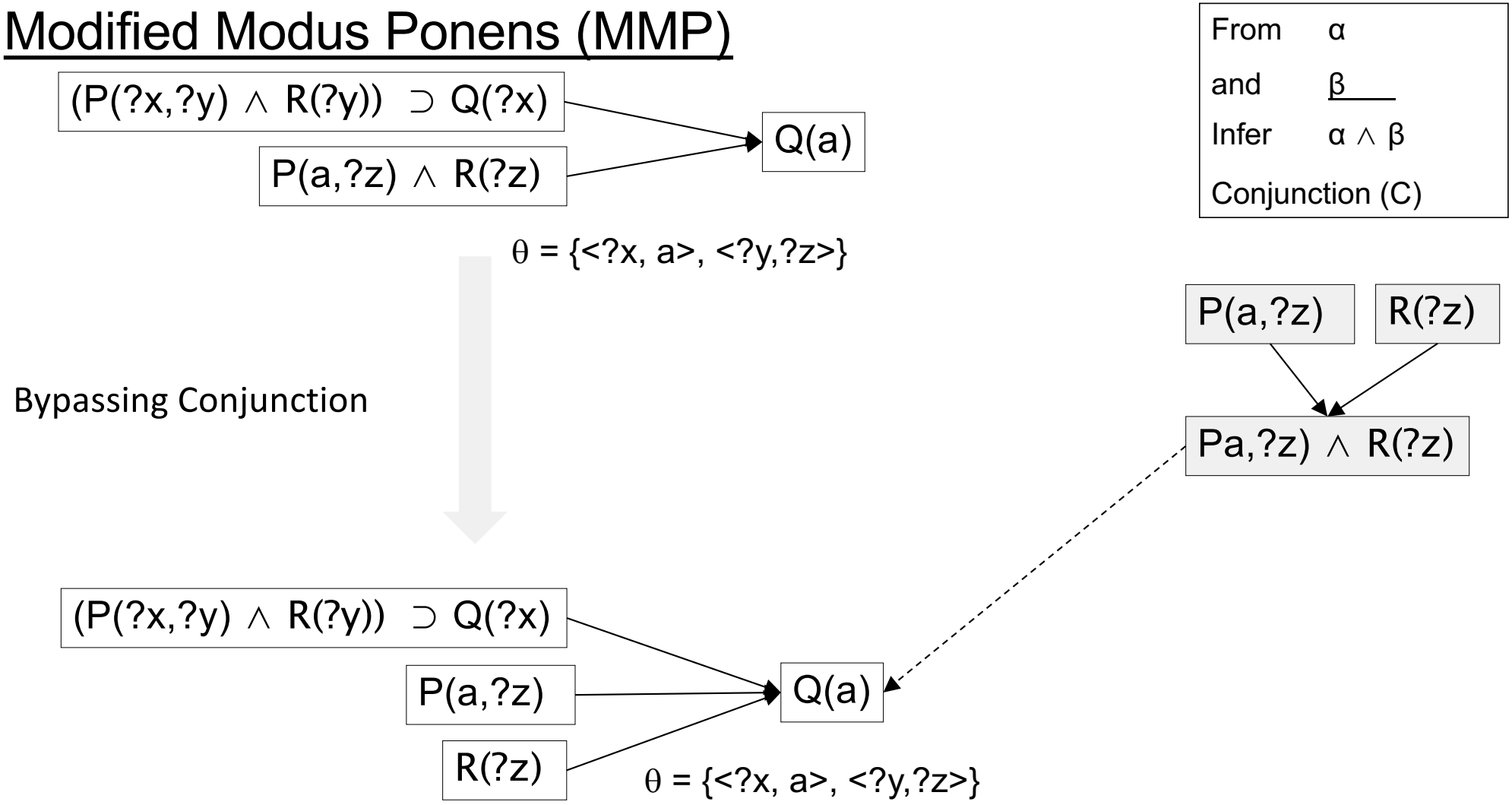
α = (Sport(tennis) $\wedge$ Likes(Alice, tennis))

$\beta \supset \delta$ = (Sport(?y) $\wedge$ Likes(?x, ?y)) $\supset$ Watches(?x, ?y)

then α unifies with β
with the substitution θ = {<?x, Alice>, <?y, tennis>}
and one can *infer*

$\delta\theta$ = Watches(?x, ?y)θ = Watches(Alice, tennis)

Modified Modus Ponens (MMP)



A shorter proof with Modified Modus Ponens

1. likes(Alice, Math) $\wedge$ likes(Alice, stories)
2. likes(?x, Math) $\supset$ likes(?x, Algebra)
3. (likes(?x, Algebra) $\wedge$ likes(?x, Physics)) $\supset$ goesTo(?x, College)
4. $\neg$likes(Alice, stories) $\vee$ likes(Alice, Physics)
5. $\neg$likes(Alice, Chemistry) $\wedge$ $\neg$likes(Alice, History)

6. likes(Alice, Math)	1, simplification
7. likes(Alice, stories)	1, simplification
8. likes(Alice, Algebra)	6, 2, MPP
9. likes(Alice, Physics)	4, 7, disjunctive syllogism
10. goesTo(Alice, College)	3, 8, 9, MPP

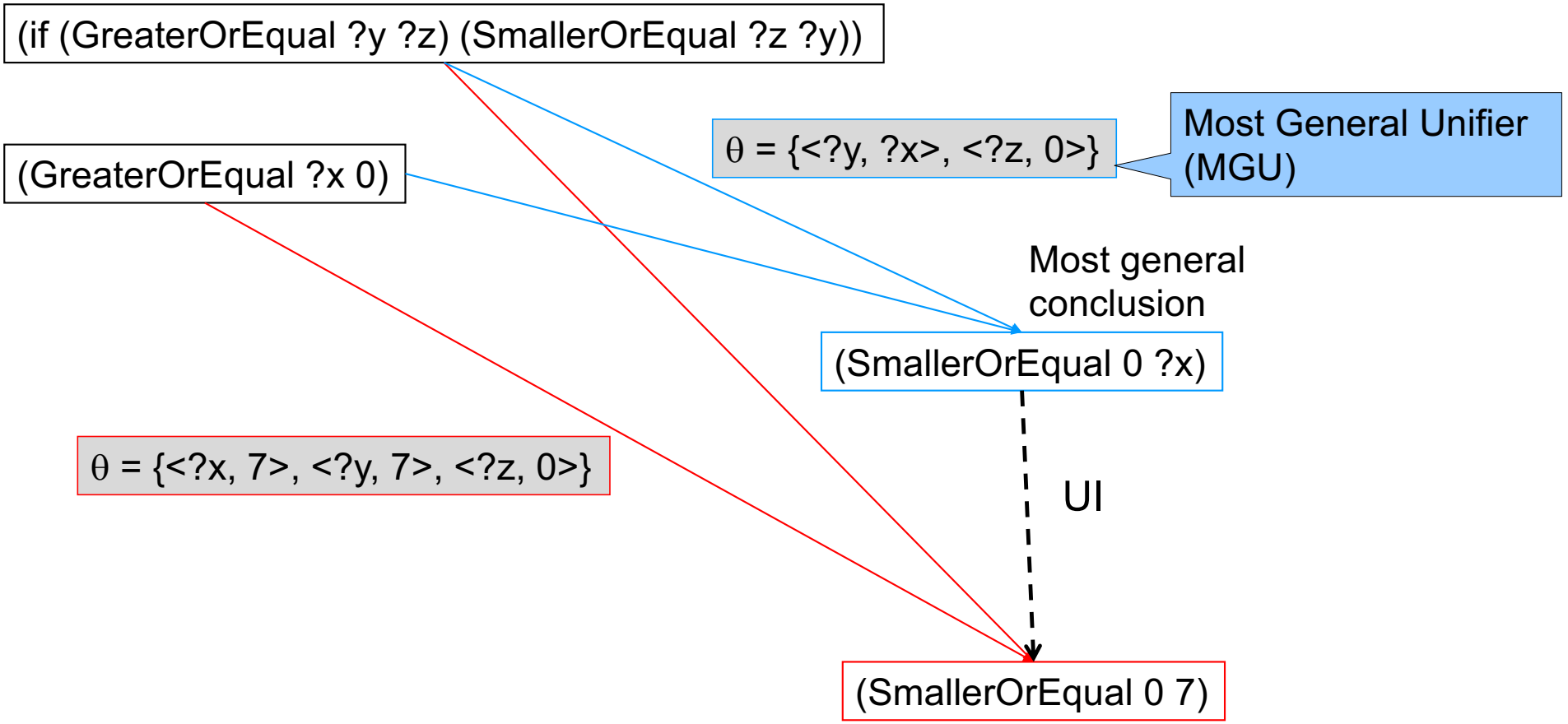
More general and more specific sentences

We say that a sentence α is *more general than* sentence β if there exists a non-empty substitution λ such that αλ = β.

Everyone loves a good student	(good-student ?x) ⊃ (loves ?y ?x)
is more general than	
Everyone's dad loves a good student	(good-student ?x) ⊃ (loves (dad ?y) ?x)
and	
Suresh loves a good a student	(good-student ?x) ⊃ (loves suresh ?x)

A more general sentence entails a less general one (generalized UI)

General Inferences and Specific Inferences



The Unification Algorithm

- The unification algorithm takes two or more formulas and finds the most general unifier for the formulas
- In the list notation for formulas there are three kinds of elements
  - lists
  - constants
  - variables
- Two constants can only unify (match) if identical
- Two lists are unified element-by-element building up the substitution as we scan the lists.
- A variable can match another variable, or a constant, or a list not containing the variable

Standardizing variables apart

Consider the two formulas

(if (GreaterOrEqual 7 ?**x**) (SmallerOrEqual ?x 7))

(GreaterOrEqual ?**x** 0)

Clearly one cannot substitute **?x** with both 0 and 7

Solution: Rename variables differently in each formula.

Standardizing variables apart

Solution: Rename variables differently in each formula.

(if (GreaterOrEqual 7 ?**x**) (SmallerOrEqual ?x 7))

(GreaterOrEqual ?**z** 0)

θ = {<?z, 7>, <?x, 0>}

… and one can now derive the conclusion (SmallerOrEqual 0 7)

The Unification Algorithm

Algorithm *Unify* returns the MGU for *arg1* and *arg2*

```
Unify(arg1, arg2)
        Return SubUnify(arg1, arg2, ( ))
```

Call an auxiliary function *SubUnify* adding a third argument.

- to build the substitution θ piece by piece
- initially θ is the empty list

Algorithm SubUnify (arg1, arg2, θ)

1. If *arg1* and *arg2* are both constants then they must be equal (else return *NIX*)
2. If *arg1* is a variable, call *VarUnify(arg1, arg2, θ)*
3. If *arg2* is a variable, call *VarUnify(arg2, arg1, θ)*

   /* at this point one or both must be lists */

4. If *Length(arg1) ≠ Length(arg2)* return *NIX*
5. For each corresponding element in *arg1* and *arg2*
   Call *SubUnify* recursively building up the substitution θ

Algorithm VarUnify(var, arg, θ)

1. If *var* occurs* in *arg* return *NIX*
2. If *var* has a value <*var*, *pat*> in θ
   return *SubUnify(pat, arg)*
3. If *var* = *arg* return θ
4. Augment $\theta \leftarrow \{<var, arg>\} \cup \theta$ and return θ

*Occurs check: Should not be able to unify *?x* with *(plus ?x 1),* for example

Unification: an example

α = (and (Sport tennis) (Likes Alice tennis))
β ⊃ δ = (if (and (Sport ?y) (Likes ?x ?y)) (Watches ?x ?y))

Unify(α, β) → SubUnify(α, β, ( ))

Both α and β are lists of length 3 → Call SubUnify with each element

1. SubUnify(and, and, ( )) → identical constants → θ = ( )

2. SubUnify((Sport tennis), (Sport ?y), ( )) → both are lists of length 2 → 2 recursive calls
   1. SubUnify(Sport, Sport, ( )) → identical constants → θ = ( )
   2. SubUnify(tennis, ?y, ( )) → Call VarUnify(?y, tennis, θ) → θ = (<?y, tennis>)

3. SubUnify((Likes Alice tennis), (Likes ?x, ?y), (<?y, tennis>)) → both are lists of length 3 → 3 recursive calls
   1. SubUnify(Likes, Likes, (<?y, tennis>)) → identical constants → θ = (<?y, tennis>)
   2. SubUnify(Alice, ?x, (<?y, tennis>)) → Call VarUnify(?x, Alice, θ) → θ = (<?y, tennis>, <?x, Alice>)
   3. SubUnify(tennis, ?y, (<?y, tennis>, <?x, Alice>))
      → Call VarUnify(?y, tennis, θ) → ?y has a value <?y, tennis) ∈ θ
      → SubUnify(tennis, tennis, (<?y, tennis>, <?x, Alice>)) → identical constants

Return θ = (<?y, tennis>, <?x, Alice>)

Unification: another example

From α = (not (CanSee ?z ?z))
and β ⊃ δ = (if (not (CanSee ?x (feet ?x)) (MustExercise ?x))
Can we conclude (MustExercise ?x)?

Unify(α, β) → SubUnify(α, β, ( ))

Both α and β are lists of length 2 → Call SubUnify with each element

1. SubUnify(not, not, ( )) → identical constants → θ = ( )

2. SubUnify(CanSee ?z ?z), (CanSee ?x (feet ?x)), ( )) → both are lists of length 3
   → 3 recursive calls
   1. SubUnify(CanSee, CanSee, ( )) → identical constants → θ = ( )
   2. SubUnify(?z, ?x, ( )) → Call VarUnify(?z,?x, θ) → θ = (<?z,?x>)
   3. SubUnify(?z, (feet ?x), (<?z,?x>)) → Call VarUnify(?z, (feet ?x), (<?z,?x>))
      → ?z has a value <?z, ?x) ∈ θ
      → SubUnify(?x, (feet ?x), (<?z,?x>)) → Occurs check ?x ∈ (feet ?x)
      --> return NIX

Return NIX

Skolemization: existentially quantified variables

When the existential quantifier is *not* in the scope of any universal quantifier, then the variable it quantifies is replaced by a Skolem constant.

For example the statements,

(exists (z) (and (Student z) (Bright z)))

$\exists z(\text{Student}(z) \wedge \text{Bright}(z))$

(exists (y) (and (Girl y) (forall (x) (if (Boy x) (Likes x y)

$\exists y(\text{Girl}(y) \wedge \forall x(\text{Boy}(x) \supset \text{Likes}(x, y)))$

are skolemized as,

(and (Student sk1) (Bright sk1))

$(\text{Student}(sk1) \wedge \text{Bright}(sk1))$

(and (Girl sk2) (if (Boy ?x) (Likes ?x sk2)))

$((\text{Girl sk2}) \wedge ((\text{Boy ?x}) \supset (\text{Likes ?x, sk2})))$

Skolemization: existential variables within universal quantifiers

When the existential quantifier is in the scope of one or more universal quantifiers then the existentially quantified variable is a Skolem function of the corresponding universally quantified variables.

For example the statements,

(forall (x y) (exists (z) (and (LessThan x z) (LessThan y z))))

$\forall x \, \forall y \, \exists z \, (\text{LessThan}(x, z) \wedge \text{LessThan}(y, z))$

(forall (x) (if (Boy x) (exists (y) (and (Girl y) (Likes x y)))))

$\forall x \, (\text{Boy}(x) \supset \exists y(\text{Girl}(y) \wedge \text{Likes}(x, y)))$

are skolemized as,

(and (LessThan ?x (sk57 ?x ?y)) (LessThan ?y (sk57 ?x ?y)))

LessThan(?x sk57(?x ?y)) ∧ LessThan(?y sk57(?x ?y))

(if (Boy ?x) (and (Girl (sk16 ?x)) (Likes ?x (sk16 ?x))))

(Boy ?x) ⊃ (Girl(sk16 ?x) ∧ Likes(?x (sk16 ?x))))

FOL: Rules of Substitution

Recap

The following rules of substitution are also useful,

Moving a negation operator inside changes the quantifier

$$\neg\forall x\ \alpha \equiv \exists x\ \neg\alpha \quad \text{DeMorgan's law}$$

$$\neg\exists x\ \alpha \equiv \forall x\ \neg\alpha \quad \text{DeMorgan's law}$$

Two quantifiers of the same type are commutative

$$\forall x\ \forall y\ \alpha \equiv \forall y\ \forall x\ \alpha$$

$$\exists x\ \exists y\ \alpha \equiv \exists y\ \exists x\ \alpha$$

The real nature of a variable

Whether a variable is universally quantified or existentially quantified has to be decided carefully. One must keep in mind that a negation sign influences the nature of the quantifier.

Consider the formalization of "*An immortal man does not exist*"

$$\neg\exists x(\text{Man}(x) \wedge \neg\text{Mortal}(x))$$

What is the nature of the variable x?

On the surface it is bound by an existential quantifier so one might mistakenly skolemize it as $\neg((\text{Man sk11}) \wedge \neg(\text{Mortal sk11}))$ but that only talks of a specific, albeit unspecified, individual or individuals. The correct way to skolemize a formula is to first push the negation sign inside. That gives us the form,

$$\forall x\neg(\text{Man}(x) \wedge \neg\text{Mortal}(x))$$

which is another way of saying that *all men are mortal*.

The real nature of a variable

The following sentence reads "*If there exists a number that is even and odd then the Earth is flat*" and is formalized as,

$$\exists x(\text{Number}(x) \wedge \text{Even}(x) \wedge \text{Odd}(x)) \supset \text{Flat}(\text{Earth})$$

However if we rewrite the equivalent formulas as,

$$\neg(\exists x(\text{Number}(x) \wedge \text{Even}(x) \wedge \text{Odd}(x))) \vee \text{Flat}(\text{Earth})$$

$$\equiv \quad \forall x(\neg(\text{Number}(x) \wedge \text{Even}(x) \wedge \text{Odd}(x))) \vee \text{Flat}(\text{Earth})$$

$$\equiv \quad \forall x(\neg(\text{Number}(x) \wedge \text{Even}(x) \wedge \text{Odd}(x)) \vee \text{Flat}(\text{Earth}))$$

$$\equiv \quad \forall x((\text{Number}(x) \wedge \text{Even}(x) \wedge \text{Odd}(x)) \supset \text{Flat}(\text{Earth}))$$

We can see that *x is really a universally quantified variable.*

The real nature of a variable

The following example that asserts "*A detective who has a sidekick is successful*" also illustrates the point that a quantifier in the antecedent part of an implication statement is masquerading as the other quantifier. Variable y is universal.

$\forall x\ [(\text{Detective}(x) \wedge \exists y\ \text{Sidekick}(y,x)) \supset \text{Successful}(x)]$

$\equiv \quad \forall x\ [\neg(\text{Detective}(x) \wedge \exists y\ \text{Sidekick}(y,x)) \vee \text{Successful}(x)]$

$\equiv \quad \forall x\ [(\neg\text{Detective}(x) \vee \neg\exists y\ \text{Sidekick}(y,x)) \vee \text{Successful}(x)]$

$\equiv \quad \forall x\ [(\neg\text{Detective}(x) \vee \forall y\ \neg\text{Sidekick}(y,x) \vee \text{Successful}(x)]$

$\equiv \quad \forall x\ [\forall y\ (\neg\text{Detective}(x) \vee \neg\text{Sidekick}(y,x) \vee \text{Successful}(x)]$

$\equiv \quad \forall x\ [\forall y\ (\neg(\text{Detective}(x) \wedge \text{Sidekick}(y,x)) \vee \text{Successful}(x)]$

$\equiv \quad \forall x\ \forall y\ [(\text{Detective}(x) \wedge \text{Sidekick}(y,x)) \supset \text{Successful}(x)]$

Inference with a Skolem constant

In the unification algorithm the Skolem constants are simply treated as constants.

From	$\exists x$ Even(x)
And	$\forall x$ (Even(x) $\supset \neg$Odd(x))
Infer	$\exists x \neg$Odd(x)

When we skolemize the premises we get,

Even (SomeEvenNumber)
Even (?x) $\supset \neg$Odd(?x)

With the substitution {?x= SomeEvenNumber } we can infer

$\neg$Odd(SomeEvenNumber).

A constant can also be thought of as a function of arity 0.

Inference with a Skolem function

In the unification algorithm the Skolem functions are simply treated as functions.

From $\forall x \exists y$ Loves(x,y) Everyone loves someone
And $\forall x \forall y$ (Loves(x,y) $\supset$ CaresFor(x,y))
If someone loves somebody then they care for them

Infer $\forall x \exists y$ CaresFor(x,y) Everyone cares for someone

When we skolemise the premises we get,

Loves (?x (sk7(?x)))
Loves (?z ?y) $\supset$ CaresFor (?z ?y)

Applying the substitution {?z=?x, ?y=(sk7(?x))} we get the conclusion,

CaresFor (?x (sk7(?x)))

Rule Based Expert Systems

Recap

In the 1980's the idea that you can capture the knowledge of a human expert in the form of rules led to the development of Expert Systems. Rule Based Systems or *Production Systems* have been used in general to decompose a problem and address it in parts. In its most abstract form a rule or a production is a statement of the form,

Left Hand Side → Right Hand Side

in which the computation flows from the left hand side to the right hand side, that is Forward Chaining

Forward Chaining Rule Based Systems

Recap

Productions or rules can be used both in a forward direction and backward direction. In the forward direction it is in a *data driven* manner. The production then looks like,

Pattern → Action

where the pattern is in the given database. Thus a rule based system looks at a part of a state, and triggers some action when a pattern is matched. Usually the actions are to make some changes in the database describing the state.

An example of an OPS5 rule

Recap

One could write a rule to sort an array of numbers as follows

```
(p interchange
        (array ^index i ^value N)
        (array ^index {j > i} ^value {M < N}
→
        (modify 1 ^value M)
        (modify 2 ^value N))
```

Actions
Not just deduction
Basis of a programming language

We have used above the notation of the language *OPS5* (Forgy, 1981), one of the first rule based languages developed at Carnegie Mellon University.

```
(rule interchange
        IF      there is an element at index i with value N,
          AND there is an element at index j > i with value M < N
        THEN
                modify array(i) to hold M,
          AND modify array(j) to hold N)
```

XCON

Originally called R1[1] the *XCON* system was a forward chaining rule based system to help automatically configure computer systems (McDermott, 1980a; 1980b). *XCON* (for eXpert CONfigurer) was built for the computer company Digital Equipment Corporation, and helped choose components for their VAX machines. *XCON* was implemented in the rule based language *OPS5*. By 1986 *XCON* had been used successfully at DEC processing over 80,000 orders with an accuracy over 95%.

XCON is a forward chaining rule based system that worked from requirements towards configurations, without backtracking. It needed two kinds of knowledge (Jackson, 1986),

- knowledge about components, for example voltage, amperage, pinning-type and number of ports, and
- knowledge about constraints, that is, rules for forming partial configurations of equipment and then extending them successfully.

[1] McDermott's 1980 paper on R1 won the AAAI Classic Paper Award in 1999. According to legend, the name of R1 comes from McDermott, who supposedly said as he was writing it, "*Three years ago I wanted to be a knowledge engineer, and today I are one*." - http://en.wikipedia.org/wiki/Xcon

XCON: Component Knowledge

XCON stored the component knowledge in a separate database, and used its production system architecture to reason about the configuration. The following is an example of a record that describes a disk controller.

RK611*

CLASS:	UNIBUS MODULE
TYPE:	DISK DRIVE
SUPPORTER:	YES
PRIORITY LEVEL:	BUFFERED NPR
TRASFER RATE:	12 ...

XCON: Rules

Constraints knowledge is specified in the form of rules. The LHS describes patterns in partial configurations that can be extended, and the RHS did those extensions. The following is an English translation of an *XCON* rule taken from (Jackson, 1986).

DISTRIBUTE-MB-DEVICES-3
IF the most current active context is distributing massbus devices
& there is a single port disk drive that has not been assigned to a massbus
& there is no unassigned dual port disk drives
& the number of devices that each massbus should support is known
& there is a massbus that has been assigned at least one disk drive and that should support additional disk drives
& the type of cable needed to connect the disk drive to the previous device on the disk drive is known
THEN
assign the disk drive to the massbus

Backward Chaining

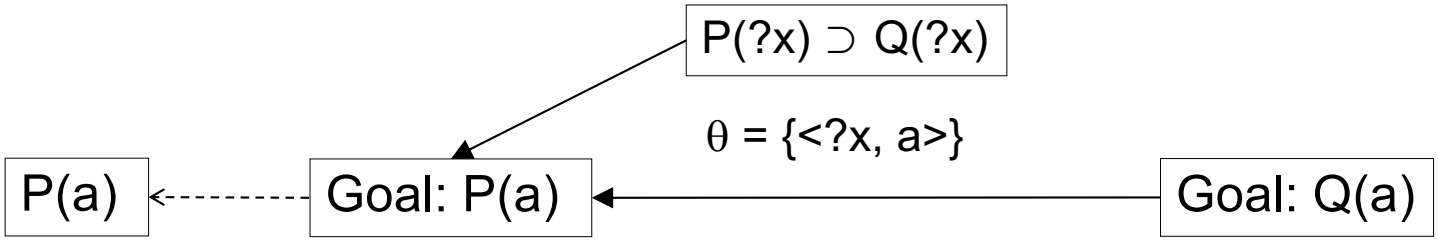
From Goals to Facts

Backward Chaining

We move from the goal to be proved towards the available facts.

From ($\alpha \supset \gamma$) and *Goal:*β* infer *Goal:*$\alpha\theta$ where θ is a unifier for γ and β and $\alpha\theta$ is the formula obtained by applying the substitution θ to α.

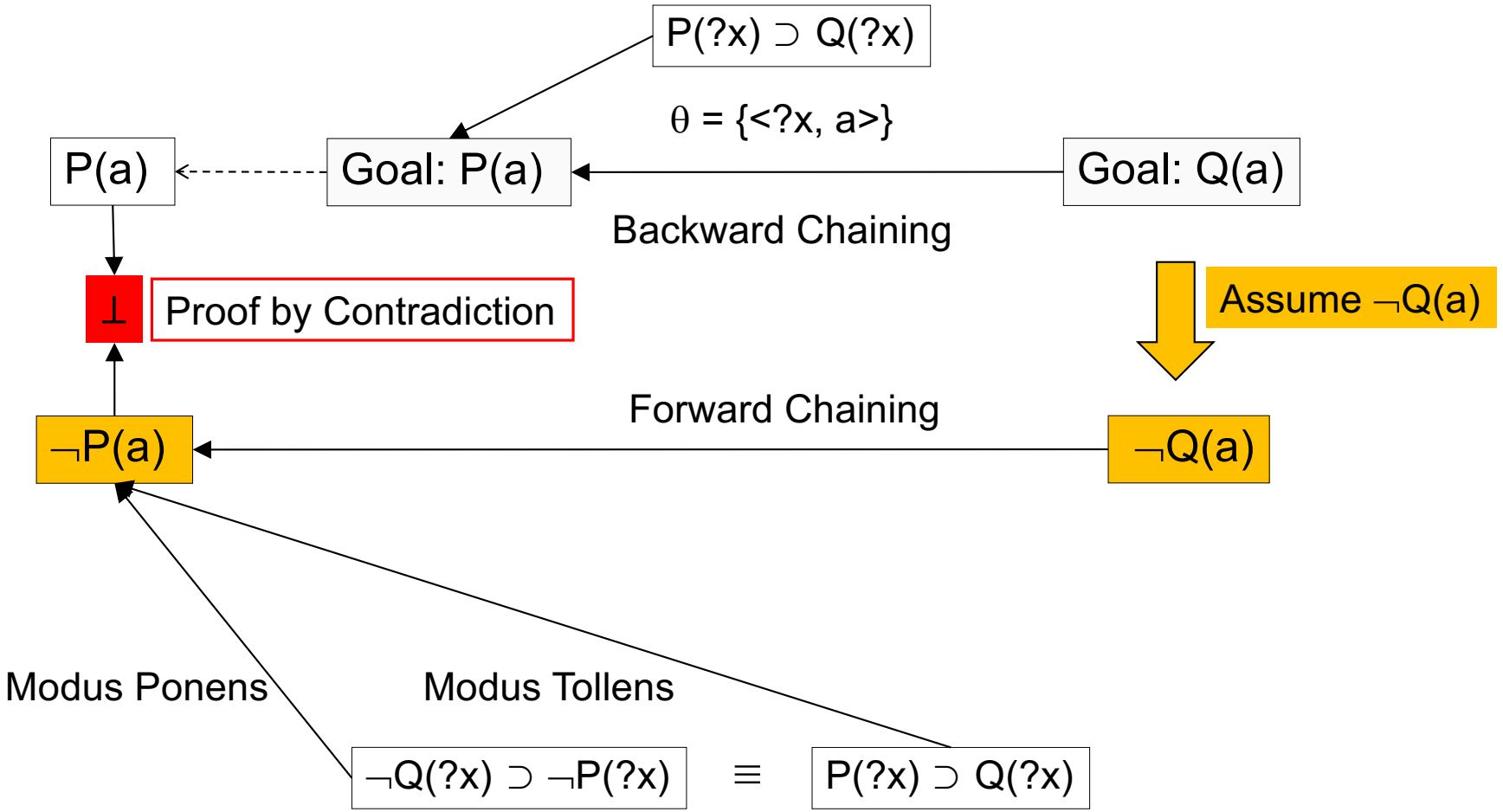
For example,



A *goal* is said to be *solved* if it matches a fact in the KB.

* The Goal: prefix essentially distinguish between fact and goal.

Backward Chaining and Modus Tollens



Backward Reasoning

- Backward reasoning is goal directed
- We only look for rules for which the consequent matches the goal.
- This results in low branching factor in the search tree
  - which rule to apply from the matching set of rules?
- Foundations of Logic Programming
  - the programming language Prolog

Deductive Retrieval

The goal need not be a *specific proposition*

It can be have variables as well

Goal formulas with variables can match facts after unification.

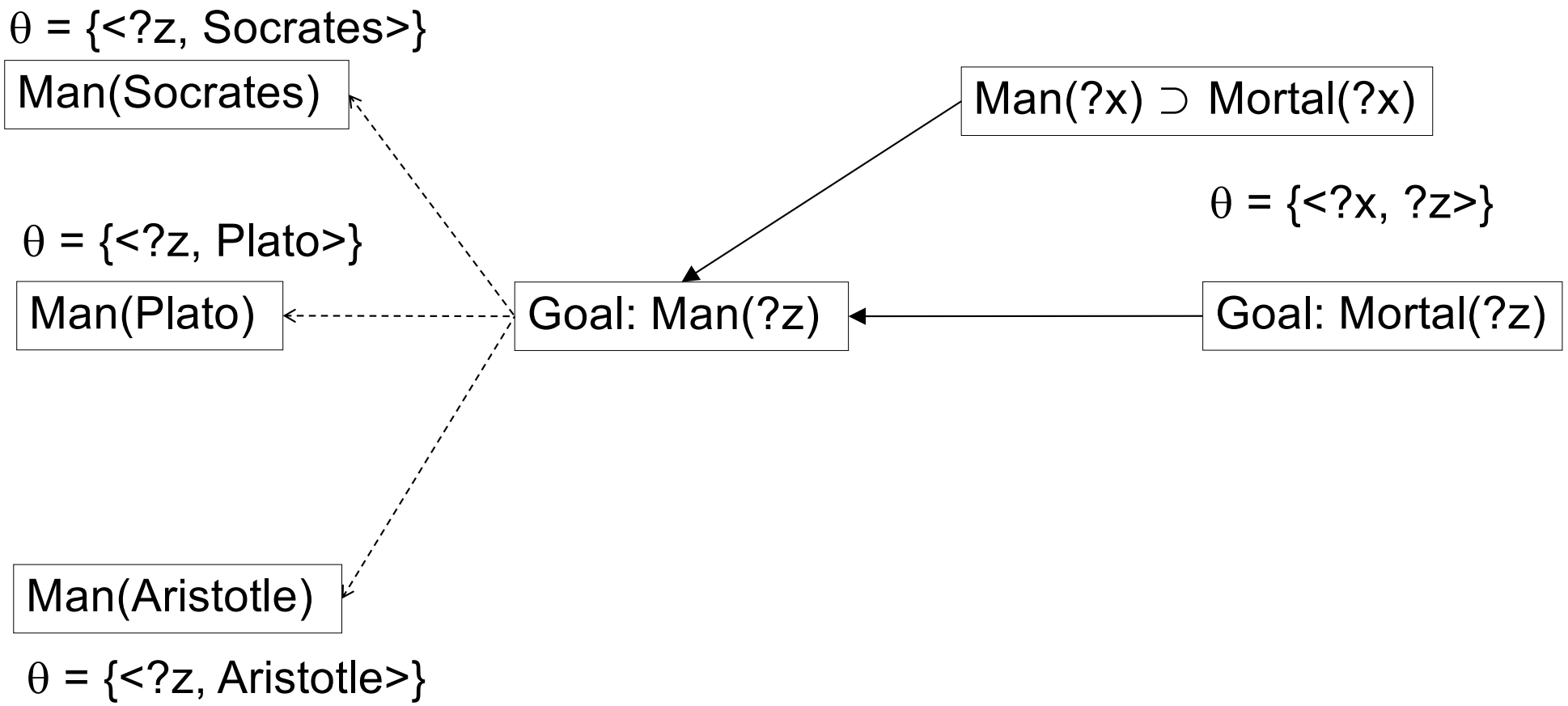
For example.

Goal: Mortal(?z) can be interpreted as an *existential* statement

Is (there a *z* such that) $\exists z$ *Mortal(z)* is true?

The answer, in addition to *yes* or *no*,
can also return a *value* for the *variable*
for which it is true.

Deductive Retrieval: possible answers



Backward Chaining (Propositional Logic)

Alice likes mathematics (P) and she likes stories (Q). If she likes mathematics (P) she likes algebra (R). If she likes algebra (R) and likes physics (S) she will go to college (T). She does not like stories (Q) or she likes physics (S). She does not like chemistry (U) and history (V).

Then the given facts are, $(P \wedge Q)$, $(P \supset R)$, $((R \wedge S) \supset T)$, $(\sim Q \vee S)$, $(\sim U \wedge \sim V)$

Equivalently

1. P
2. Q
3. $(P \supset R)$
4. $((R \wedge S) \supset T)$
5. $(Q \supset S)$
6. $\sim U$
7. $\sim V$

Goal Set (to show that)

{T} Given goal
{R, S} from 4
{P, S} from 3
{S} matches 1
{Q} from 5
{ } matches 2, success, nothing to show

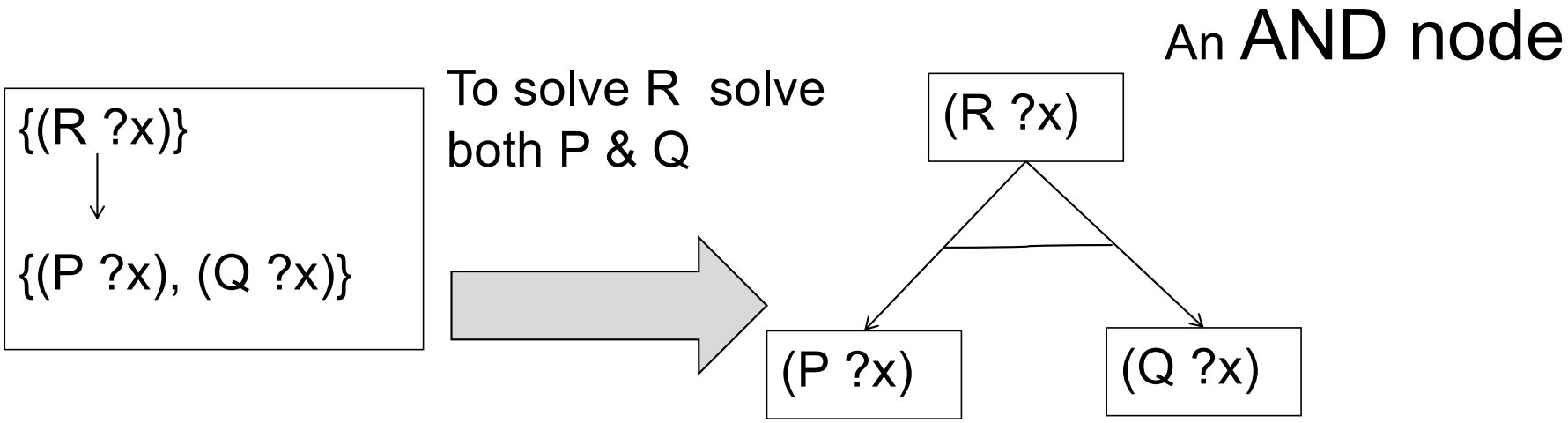
"Is T *true*?"

We answer this by backward chaining.

Backward Chaining with Conjunctive Antecedents

A goal (R ?x) and a rule (if (and (P ?x) (Q ?x)) (R ?x))

A *goal which matches the consequent* of a rule reduces to the antecedents in the rule.

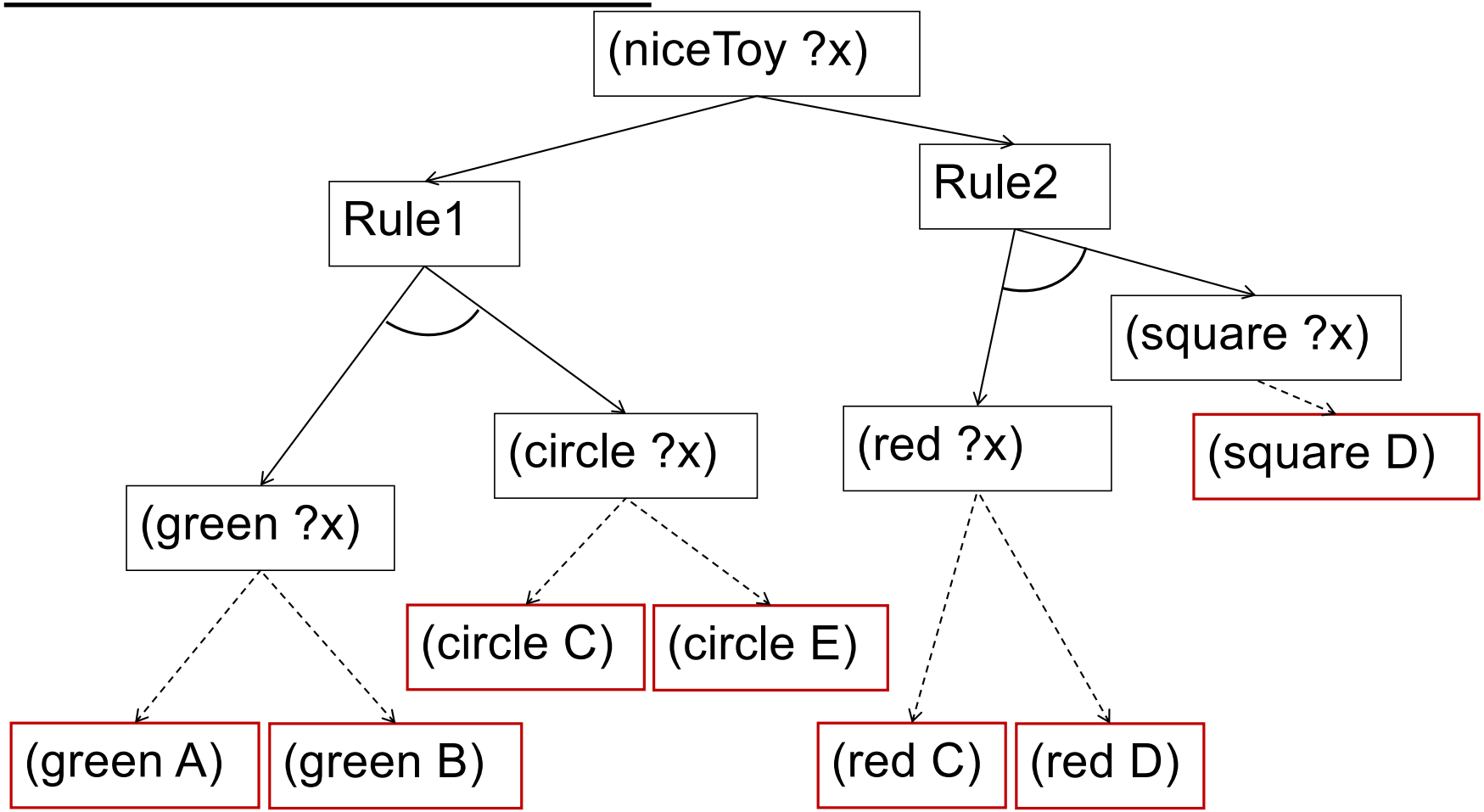


Goal Trees

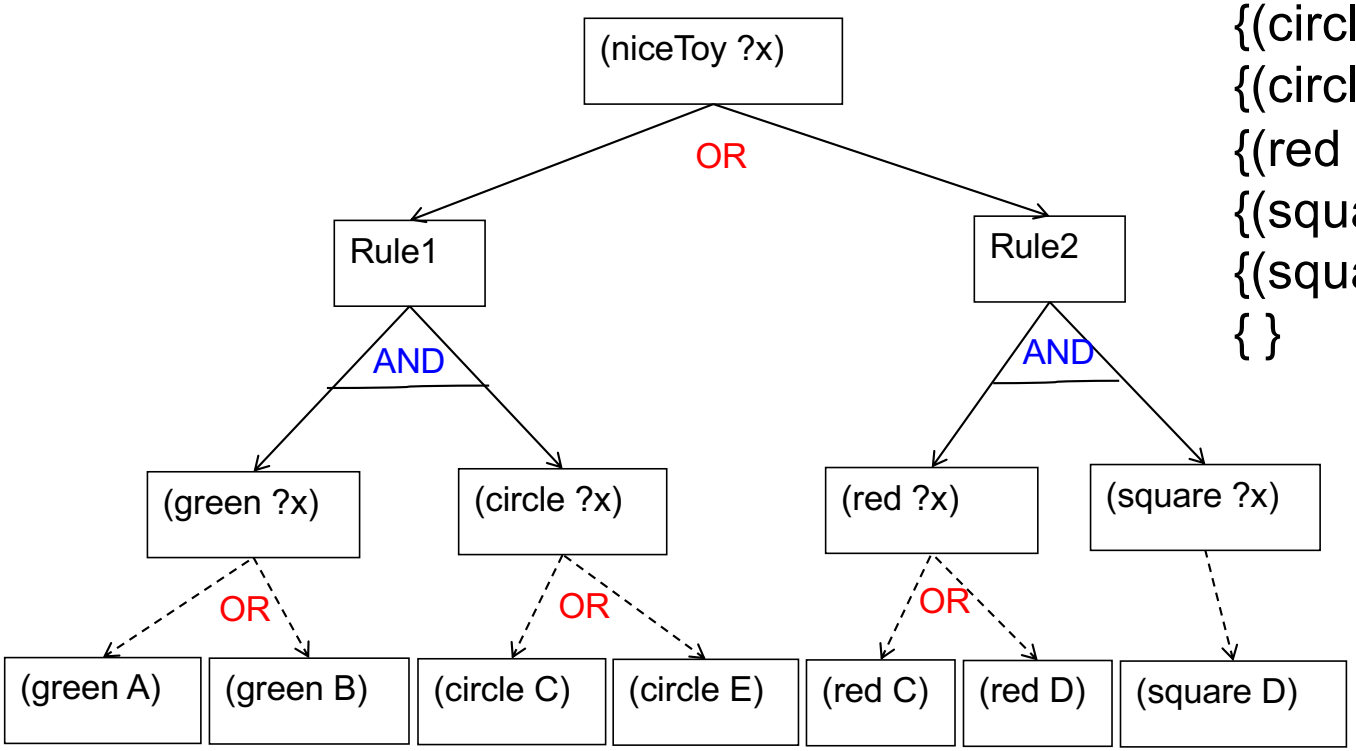
Consider the following KB in skolemized list notation, and the goal (niceToy ?z)

Rule1: (if (and (green ?x) (circle ?x)) (niceToy ?x))
Rule2: (if (and (red ?x) (square ?x)) (niceToy ?x))
(green A)
(green B)
(circle C)
(red C)
(red D)
(square D)
(circle E)

Goal tree = AND/OR tree



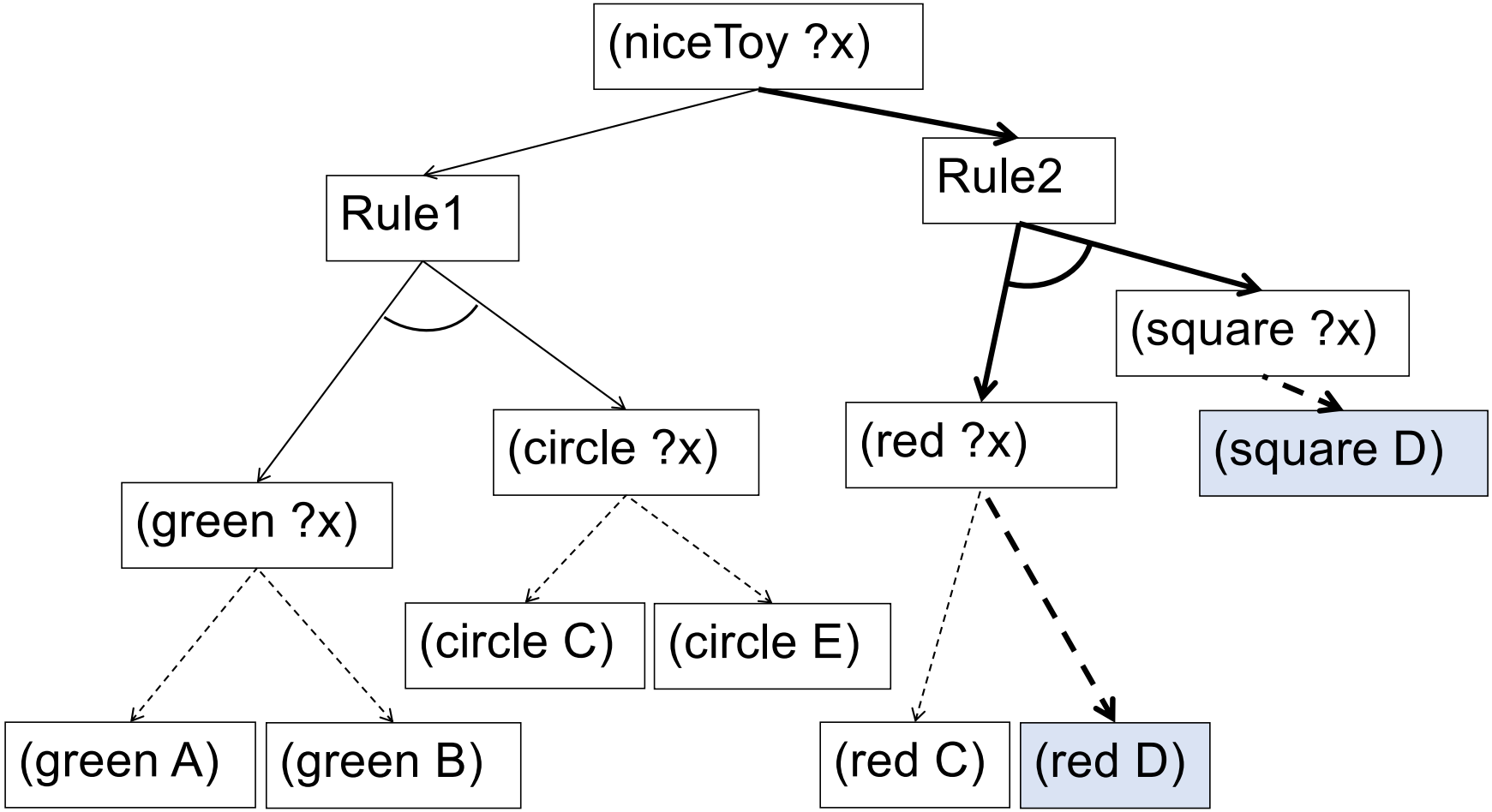
Depth First Search



Goal Set

{(niceToy ?x)}
{(green ?x), (circle ?x)} Rule1
{(circle A)} ?x=A FAIL
{(circle B)} ?x=B FAIL
{(red ?x), (square ?x)} Rule2
{(square C)} ?x=C FAIL
{(square D)} ?x=D
{ } Success

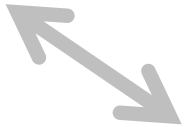
AND/OR tree: Solution = subtree



A Prolog KB (program)

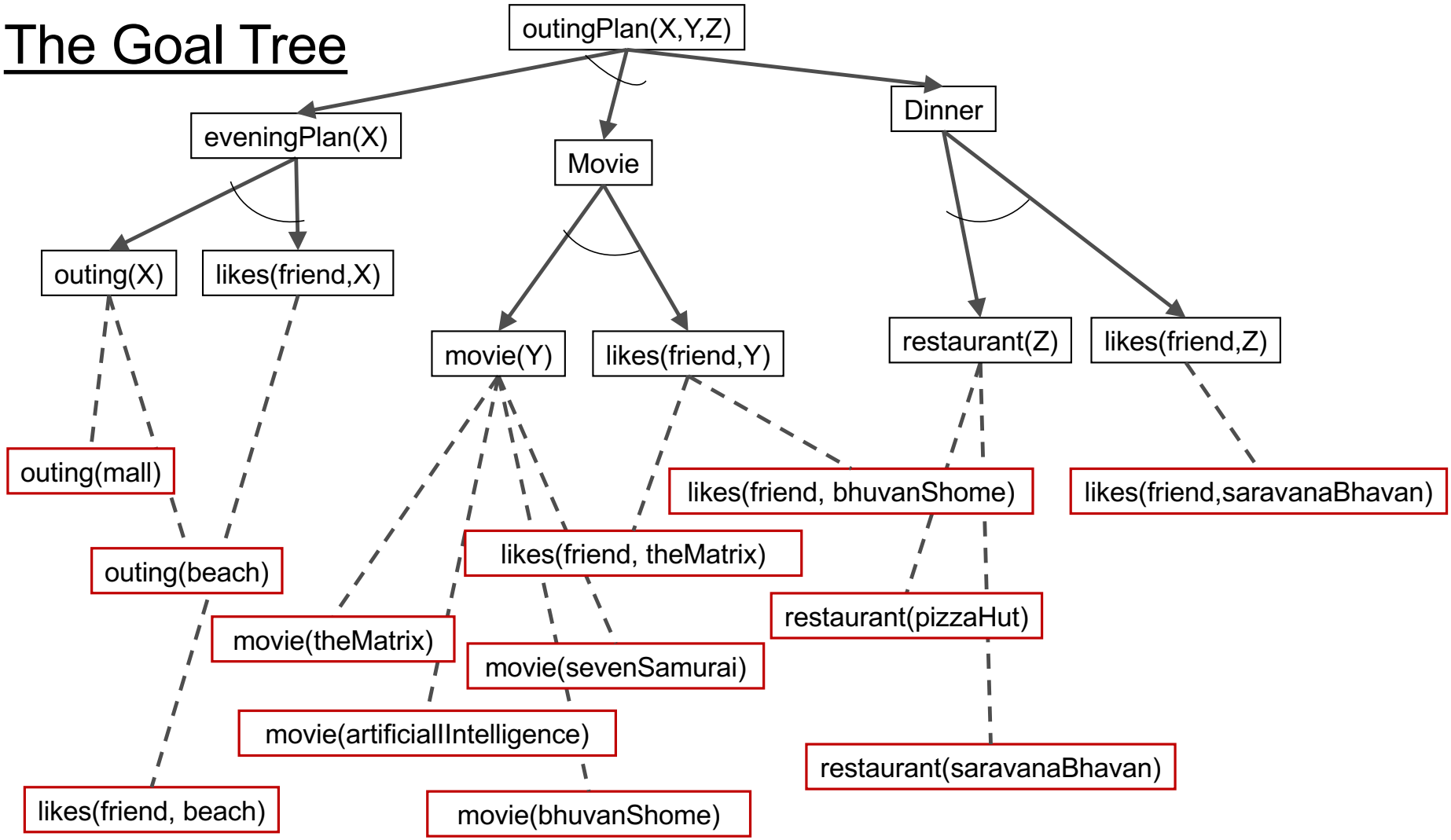
A preview

```
outingPlan(X,Y,Z) :- eveningPlan(X), moviePlan(Y), dinnerPlan(Z).
eveningPlan(X) :- outing(X), likes(friend, X).
moviePlan(X) :- movie(X), likes(friend,X).
dinnerPlan(X) :- restaurant(X), likes(friend,X).
outing(mall).
outing(beach).
movie(theMatrix).
movie(artificialIntelligence).
movie(bhuvanShome).
movie(sevenSamurai).
restaurant(pizzaHut).
restaurant(saravanaBhavan).
likes(friend, beach).
likes(friend, theMatrix).
likes(friend, bhuvanShome).
likes(friend, sarvanaBhavan).
```



```
(if (and (restaurant ?x) (likes friend ?x)) (dinnerPlan ?x))
```

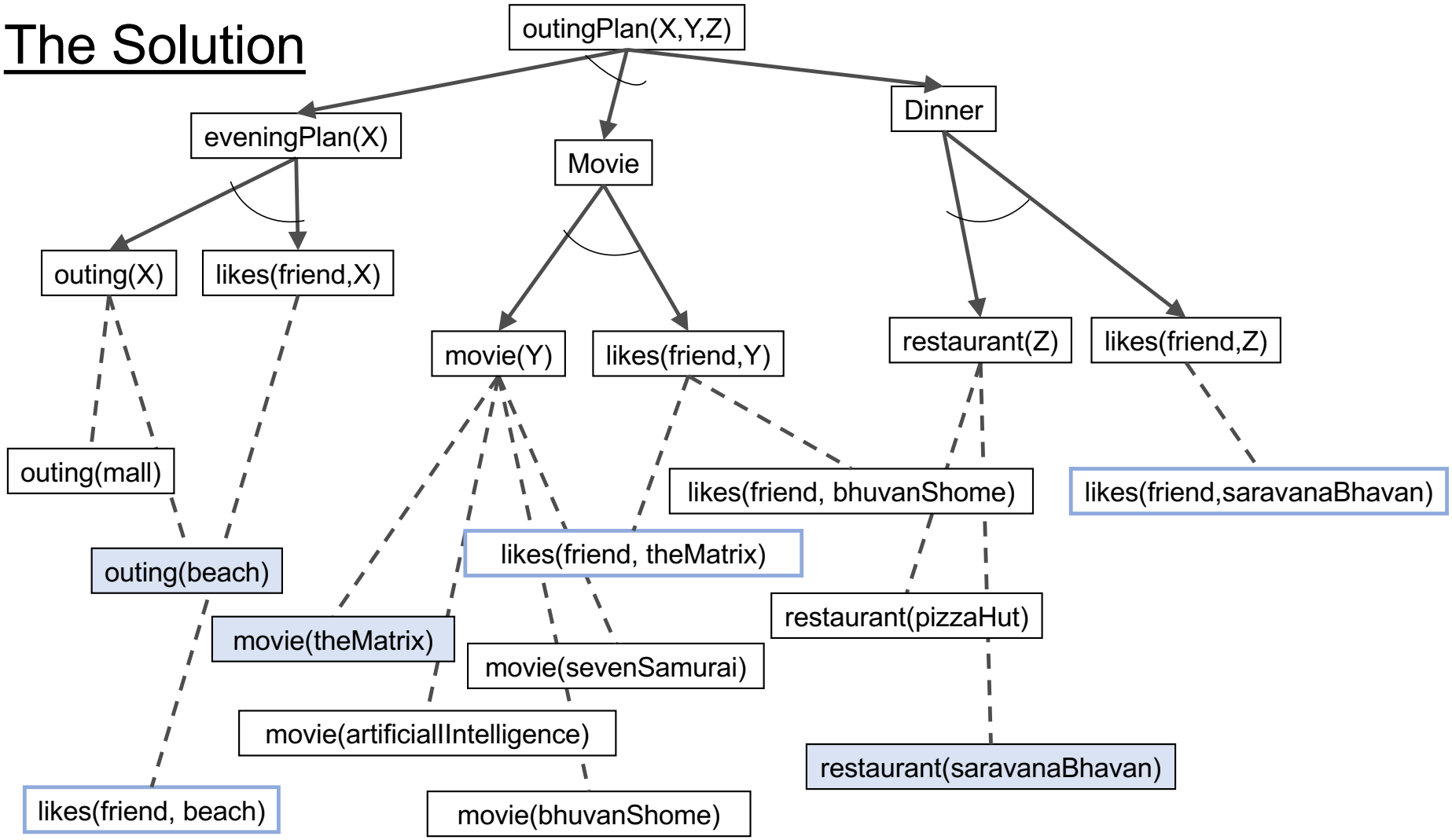
The Goal Tree



Backward Chaining: Depth First Search

{outingPlan(X,Y,Z)}	theta = { }
{eveningPlan(X), moviePlan(Y), dinnerPlan(Z)}	theta = { }
{outing(X), likes(friend, X), moviePlan(Y), dinnerPlan(Z)}	theta = { }
{likes(friend, mall), moviePlan(Y), dinnerPlan(Z)}	theta = {X=mall}
{“fail”, moviePlan(Y), dinnerPlan(Z)}	theta = {X=mall}
{outing(X), likes(friend, X), moviePlan(Y), dinnerPlan(Z)}	theta = { } backtrack
{likes(friend, beach), moviePlan(Y), dinnerPlan(Z)}	theta = {X=beach}
{moviePlan(Y), dinnerPlan(Z)}	theta = {X=beach}
{movie(Y), likes(friend,Y), dinnerPlan(Z)}	theta = {X=beach}
{likes(friend, theMatrix), dinnerPlan(Z)}	theta = {X=beach, Y=theMatrix}
{dinnerPlan(Z)}	theta = {X=beach, Y=theMatrix}
{restaurant(Z), likes(friend,Z)}	theta = {X=beach, Y=theMatrix}
{likes(friend,pizzaHut)}	theta = {X=beach, Y=theMatrix, Z=pizzaHut}
{“fail”}	theta = {X=beach, Y=theMatrix, Z=pizzaHut}
{restaurant(Z), likes(friend,Z)}	theta = {X=beach, Y=theMatrix} backtrack
{likes(friend, saravanaBhavan)}	theta = {X=beach, Y=theMatrix, Z= saravanaBhavan }
{ }	theta = {X=beach, Y=theMatrix, Z= saravanaBhavan }

The Solution



A not so easy problem

Consider the following KB

Recap

{(O A B), (O B C), (not (M A)), (M C)}

Where O is a binary predicate symbol
M is a unary predicate symbol
A, B and C are constant symbols

What is the above KB describing?

Observe that we have not used helpful names!

Remember – meaning lies in the mind of the beholder

Interpretation 1

{(O A B), (O B C), (not (M A)), (M C)}

Domain: Blocks World

Predicate symbols

O → On

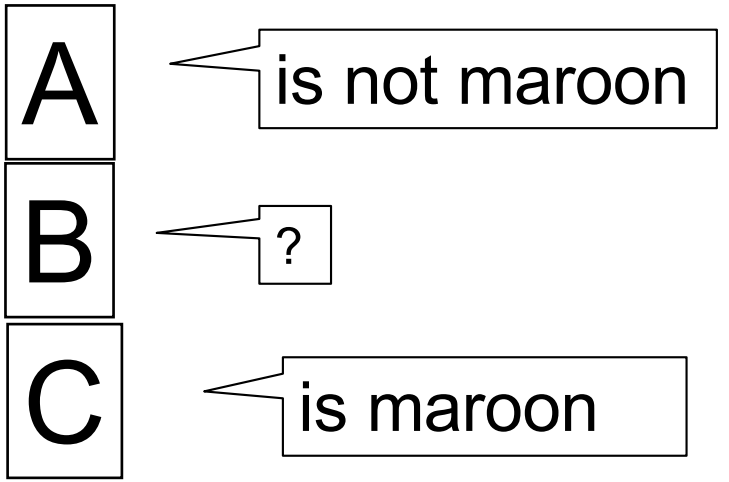
M → Maroon

Constant Symbols

A, B, C → blocks

A is on B

B is on C



Recap

Interpretation 2

{(O A B), (O B C), (not (M A)), (M C)}

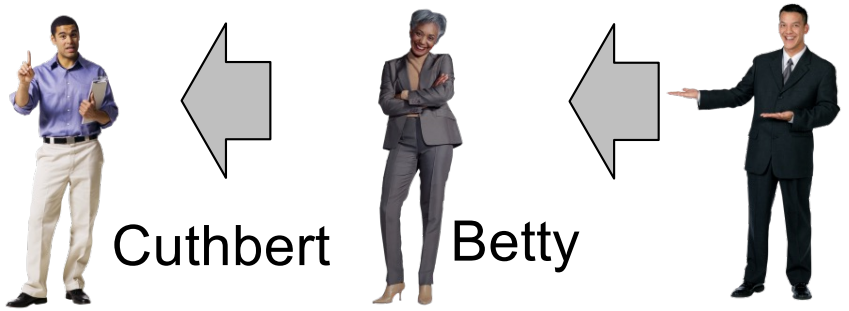
Domain: People

Predicate symbols
O → LookingAt
M → Married

Constant Symbols
A → Archie
B → Betty
C → Cuthbert

Betty is looking at Cuthbert

Archie is looking at Betty



Recap

The Goal

{(O A B), (O B C), (not (M A)), (M C)}

Given the KB and the goal

(exists (x y) (and (O x y) (not (M x)) (M y)))

or equivalently

(and (O ?x ?y) (not (M ?x)) (M ?y))

…is clearly entailed

Interpretations of the query are,

Blocks World: *Is there a not-maroon block on a maroon block?*

People: *Is a not-married person looking at a married one?*

Incompleteness of Backward and Forward Chaining

Given the KB,

{(O A B), (O B C), (not (M A)), (M C)}

And the Goal,

(and (O ?x ?y) (not (M ?x)) (M ?y))

Neither Forward Chaining nor Backward Chaining is able to generate a proof.

Both are Incomplete!

Next, we look at a proof method,
the Resolution Refutation System,
that is Sound and Complete for FOL