Knowledge Representation and Reasoning with First Order Logic

Module 2

Deepak Khemani

The Syllabus

Introduction: Overview and Historical Perspective

First Order Logic: A logic with quantified variables.

Module 1 (2 hours): Syntax, Semantics, Entailment and Models, Proof Systems, Knowledge Representation.

Module 2 (2 hours): Skolemization, Unification, Deductive Retrieval, Forward Chaining, Backward Chaining

Module 3 (2 hours): Resolution Refutation in FOL, Horn Clauses and Logic Programming

Module 4 (2 hours): Variations on FOL

Text book

Deepak Khemani. A First Course in Artificial Intelligence (Chapters 12 & 13), McGraw Hill Education (India), 2013.

Recap

Knowledge and Reasoning – necessary for intelligence

What does the agent know

and

what else does the agent know as a consequence of what it knows?

Module 2

Reasoning

Recap

The manipulation of symbols in a meaningful manner.

Maths is replete with *algorithms* we use –

- Addition and multiplication of multi-digit numbers
- Long division
- Solving systems of linear equations
- Fourier transforms, convolution…

Recap

The Syllogism

The Greek syllogism embodies the notion of **formal logic**.

An argument is valid if it conforms to a valid **form**

All men are mortal
Socrates is a man
____
Socrates is mortal

All cities are congested
Chennai is a city
____
Chennai is congested

All politicians are honest
Sambit is a politician
____
Sambit is honest

The Socratic argument

In a **valid** argument
IF the premises are true
THEN the conclusions are **necessarily** true

Some common rules of inference

Recap

From $\alpha \supset \beta$
and $\underline{\alpha}$
Infer β
Modus Ponens (MP)

From $\alpha \supset \beta$
and $\underline{\sim\beta}$
Infer $\sim\alpha$
Modus Tollens (MT)

From α
and $\underline{\beta}$
Infer $\alpha \wedge \beta$
Conjunction (C)

From $\alpha \vee \beta$
and $\underline{\sim\alpha}$
Infer β
Disjuncntive Syllogism (DS)

From $\underline{\alpha}$
Infer $\alpha \vee \beta$
Addition (A)

From $\underline{\alpha \wedge \beta}$
Infer α
Simplification (S)

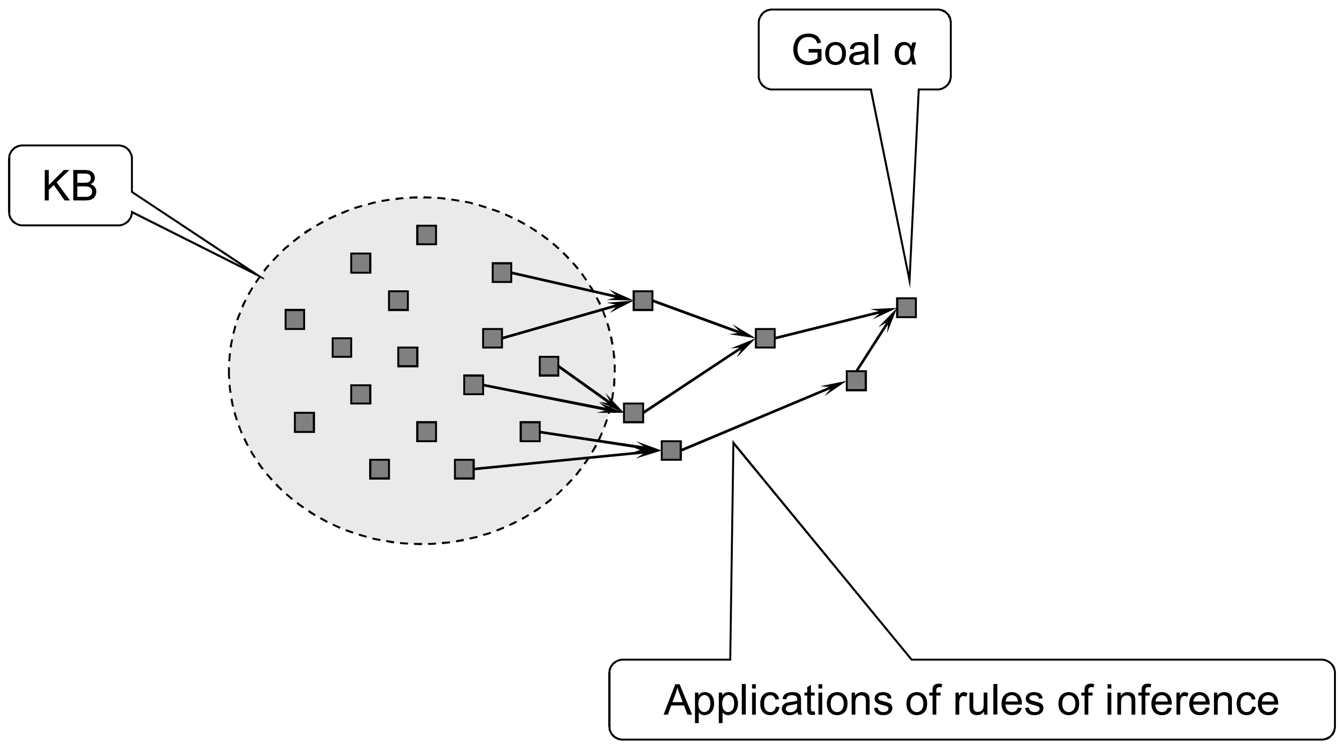
From $\alpha \supset \beta$
and $\underline{\beta \supset \gamma}$
Infer $\alpha \supset \gamma$
Hypothetical Syllogism (HS)

From $(\alpha \supset \beta) \wedge (\gamma \supset \delta)$
and $\underline{\alpha \vee \gamma}$
Infer $\beta \vee \delta$
Constructive Dilemma (CD)

From $(\alpha \supset \beta) \wedge (\gamma \supset \delta)$
and $\underline{\sim\beta \vee \sim\delta}$
Infer $\sim\alpha \vee \sim\gamma$
Destructive Dilemma (DD)

Proof

Recap



Semantics (Propositional Logic)

Recap

Atomic sentences in Propositional Logic can stand for anything. 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. She does not like stories or she likes physics. She does not like chemistry and history.

Encoding: P = Alice likes mathematics. Q = Alice likes stories. R = Alice likes algebra. S = Alice likes physics. T = Alice will go to college. U = Alice likes chemistry. V = Alice likes history.

Then the given facts are,

$(P \wedge Q)$

$(P \supset R)$

$((R \wedge S) \supset T)$

$(\sim Q \vee S)$

$(\sim U \wedge \sim V)$

If the above sentences are true is it necessarily true that "Alice will go to college"?

That is " Is T *true*?"

We answer this by producing a proof (of T)

Proofs in Propositional Logic

1.	$(P \wedge Q)$	premise
2.	$(P \supset R)$	premise
3.	$((R \wedge S) \supset T)$	premise
4.	$(\neg Q \vee S)$	premise
5.	P	1, simplification
6.	Q	1, simplification
7.	R	2, 5, modus ponens
8.	S	4, 6, disjunctive syllogism*
9.	$(R \wedge S)$	7, 8, conjunction
10.	T	3, 9, modus ponens

*Strictly speaking a substitution step $Q \equiv \neg\neg Q$ has to be applied before disjunctive syllogism is applicable.

The First Order version

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*.

FOL: Rules of Inference

Recap

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,

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

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

Examples:

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

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

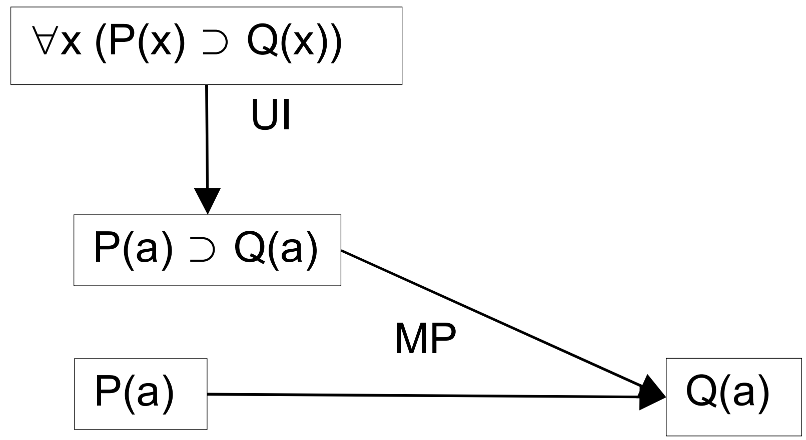
The FOL Proof

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. First a relevant instantiation of a rule is created. Then the rule instance is used by modus ponens to produce the consequent.

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

List notation

Recap

Standard mathematical notation

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

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 denoting the values to be substituted for the variables.

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,

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

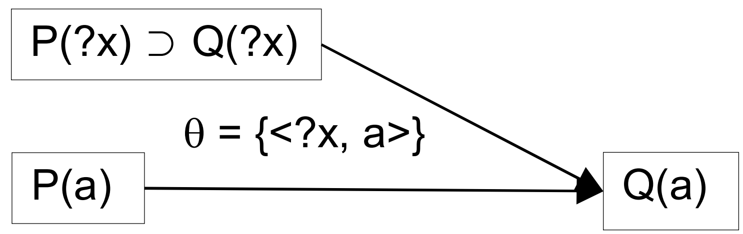
We call the common reduced form φ as the *factor*.

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,

MMP



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

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

$$\alpha \quad = (Sport(tennis) \wedge Likes(Alice, tennis))$$

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

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

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

A shorter proof with Modified Modus Ponens

1. likes(Alice, Math) ∧ likes(Alice, stories)
2. likes(?x, Math) ⊃ likes(?x, Algebra)
3. (likes(?x, Algebra) ∧ likes(?x, Physics)) ⊃ goesTo(?x, College)
4. ¬likes(Alice, stories) ∨ likes(Alice, Physics)
5. ¬likes(Alice, Chemistry) ∧ ¬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.	((likes(Alice, Algebra) ∧ likes(Alice,Physics))	9, 10, conjunction
11.	goesTo(Alice, College)	3, 10, MPP

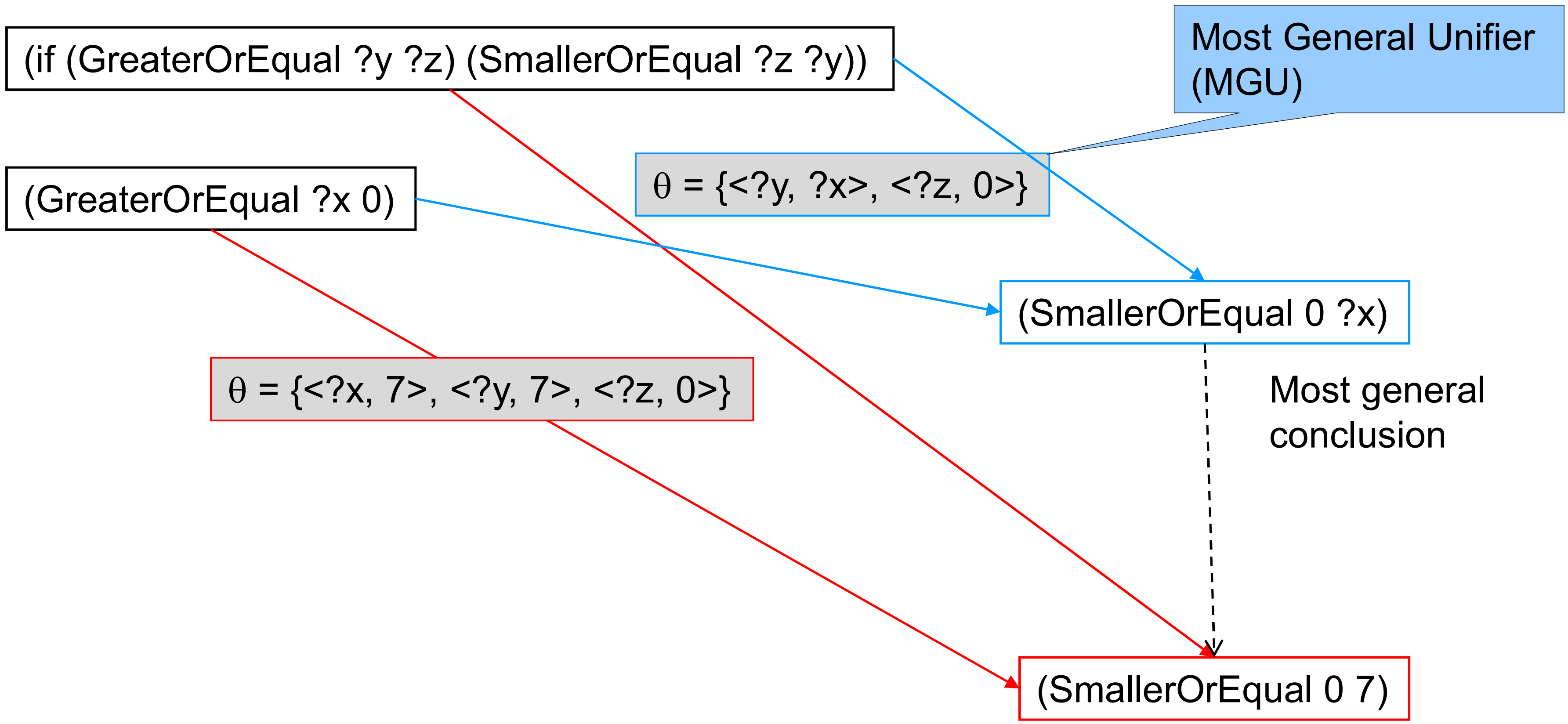
More general and more specific sentences

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

$\boxed{\text{(if (GreaterOrEqual 7 ?}\mathbf{x}\text{) (SmallerOrEqual ?x 7))}}$

$\boxed{\text{(GreaterOrEqual ?}\mathbf{x}\text{ 0)}}$

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 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* exists* 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 $\theta \leftarrow \{\langle var, arg \rangle\} \cup \theta$ and return θ

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

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))

$\exists z(Student(z) \wedge Bright(z))$

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

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

are skolemized as,

(and (Student sk1) (Bright sk1))

$(Student(sk1) \wedge Bright(sk1))$

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

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

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))))

$\forall x\ \forall y\ \exists z\ (LessThan(x, z) \land LessThan(y, z))$

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

$\forall x\ (Boy(x) \supset \exists y(Girl(y) \land Likes(x, y))$

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

$\neg\forall x\ \alpha$	$\equiv$	$\exists x\ \neg\alpha$	DeMorgan's law
$\neg\exists x\ \alpha$	$\equiv$	$\forall x\ \neg\alpha$	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*" which is another way of saying that *all men are mortal*.

$$\neg\exists x\ (\mathrm{Man}(x) \land \neg\mathrm{Mortal}(x))$$

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 $\neg$((Man sk11) $\land$ $\neg$(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,

$$\forall x \neg(\mathrm{Man}(x) \land \neg\mathrm{Mortal}(x))$$

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,

$$(\exists x\ (Number(x) \land Even(x) \land Odd(x))) \supset Flat(Earth)$$

However if we rewrite the equivalent formulas as,

$$\neg(\exists x\ (Number(x) \land Even(x) \land Odd(x))) \lor Flat(Earth)$$

$$\equiv \quad \forall x(\neg(Number(x) \land Even(x) \land Odd(x))) \lor Flat(Earth)$$

$$\equiv \quad \forall x(\neg(Number(x) \land Even(x) \land Odd(x)) \lor Flat(Earth))$$

$$\equiv \quad \forall x((Number(x) \land Even(x) \land Odd(x)) \supset Flat(Earth))$$

We can see that *x is really 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.

$\forall x\ (Detective(x) \land \exists y\ Sidekick(y,x)) \supset Successful(x))$

$\equiv \quad \forall x\ (\neg Detective(x) \lor \neg\exists y\ Sidekick(y,x) \lor Successful(x))$

$\equiv \quad \forall x\ (\neg Detective(x) \lor \forall y\ \neg Sidekick(y,x) \lor Successful(x))$

$\equiv \quad \forall x\ \forall y\ (\neg Detective(x) \lor \neg Sidekick(y,x) \lor Successful(x))$

$\equiv \quad \forall x\ \forall y\ (\neg(Detective(x) \land Sidekick(y,x)) \lor Successful(x))$

$\equiv \quad \forall x\ \forall y\ (\neg(Detective(x) \land Sidekick(y,x)) \supset Successful(x))$

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 constant

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

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

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 a rule

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))
```

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     IF 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

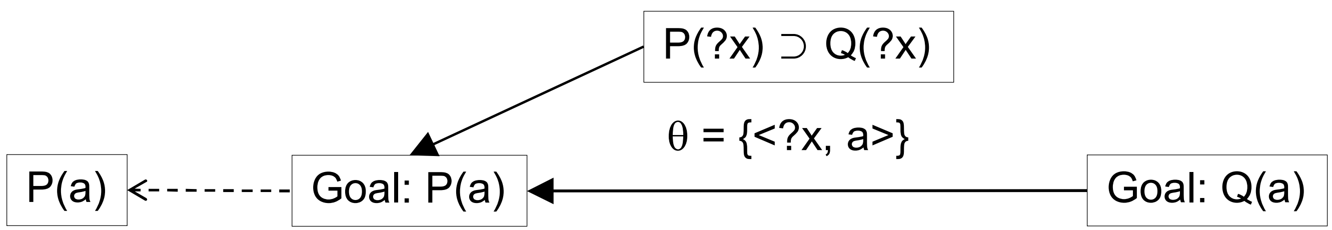
Constraints knowledge is specified in the form of rules. The LHS describes patterns in partial configurations that can be extended, and the RHS did those extensions. The following is an English translation of an *XCON* rule taken from (Jackson, 1986).

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

In Backward Chaining we move from the goal to be proved towards 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. In the above example we start with the goal of proving Q(a) and reduce to the sub-goal P(a), which is satisfied in the KB.

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

Formulas with variables can match facts.

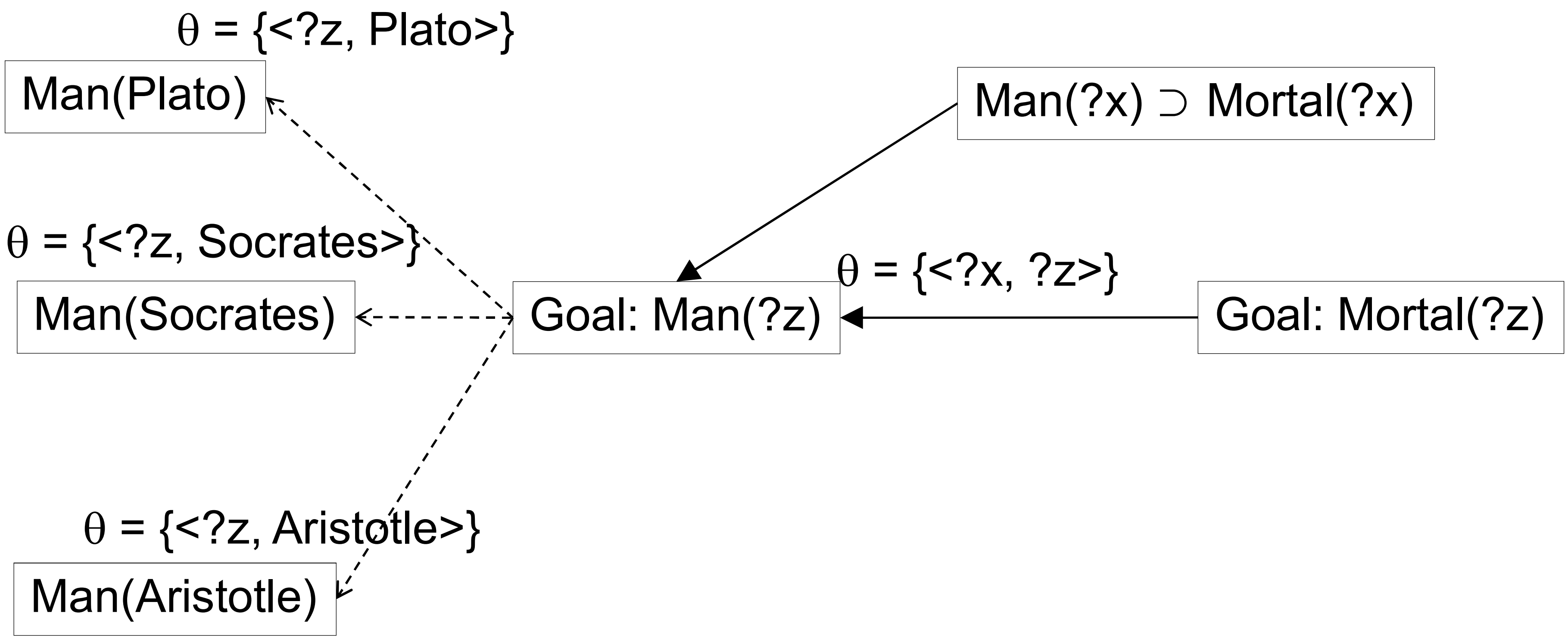
For example the goal   Goal: Mortal(?z)

can be interpreted as an existential statement

> Is the statement $\exists z$ *Mortal(z)* true?

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

Deductive Retrieval: 3 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

{T}	Given goal
{R, S}	from 4
{P, S}	from 3
{S}	1
{Q}	from 5
{ }	2, success

"Is T *true*?"

We answer this by backward chaining.

Backward Chaining with Conjunctive Antecedents

A goal (R ?x) with 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.

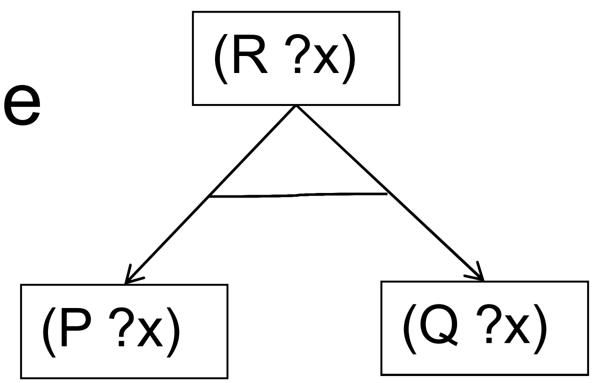
{(R ?x)}

↓

{(P ?x) (Q ?x)}

An AND node

To solve R
solve both P & Q

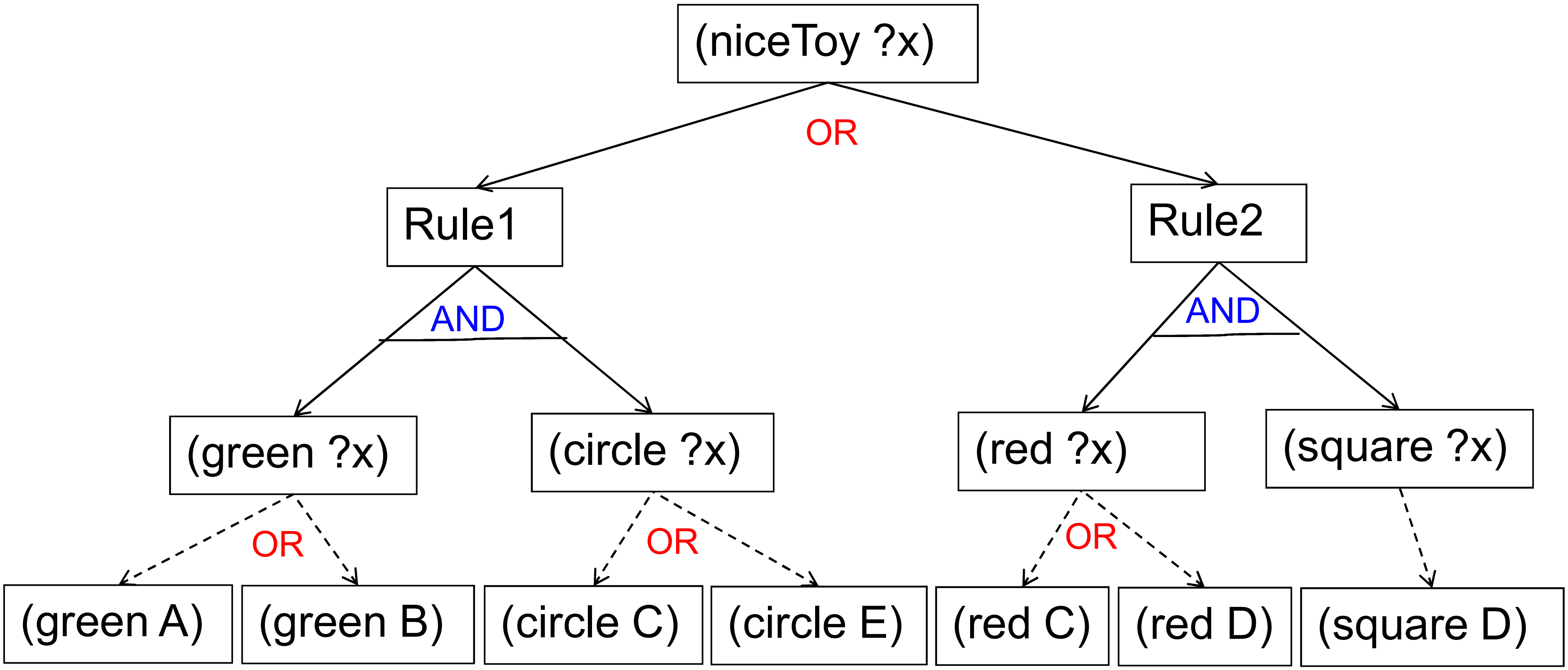


Goal Trees

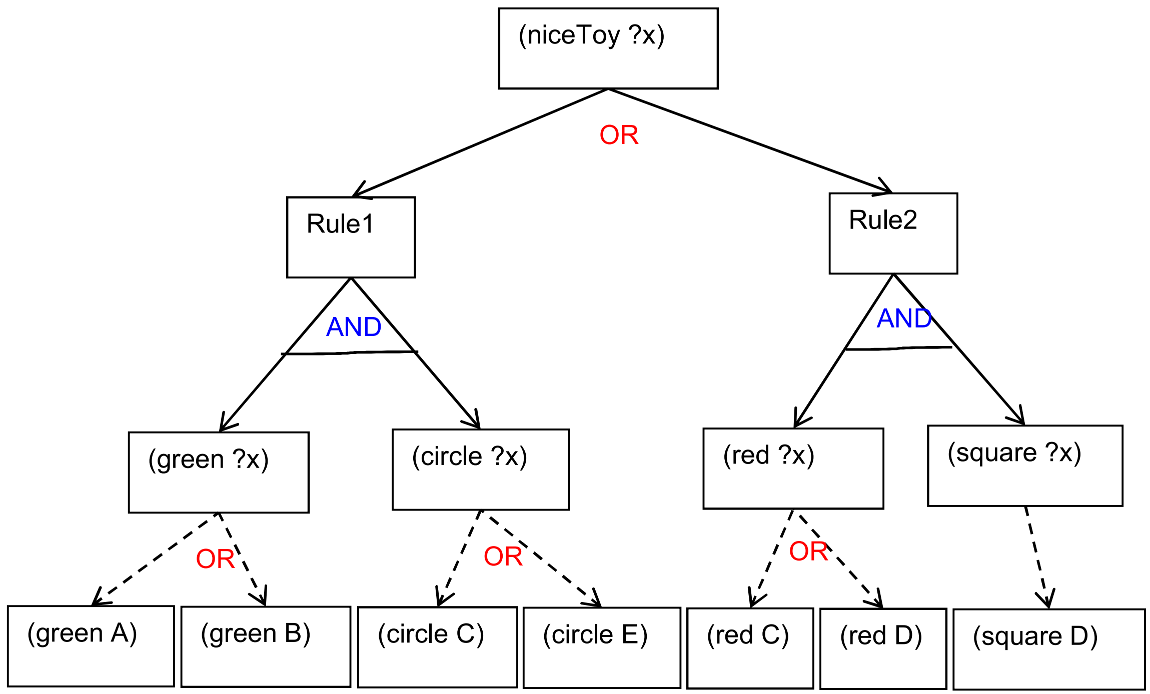
Consider the following KB in skolemized list notation, and the goal (niceToy ?z)

```
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



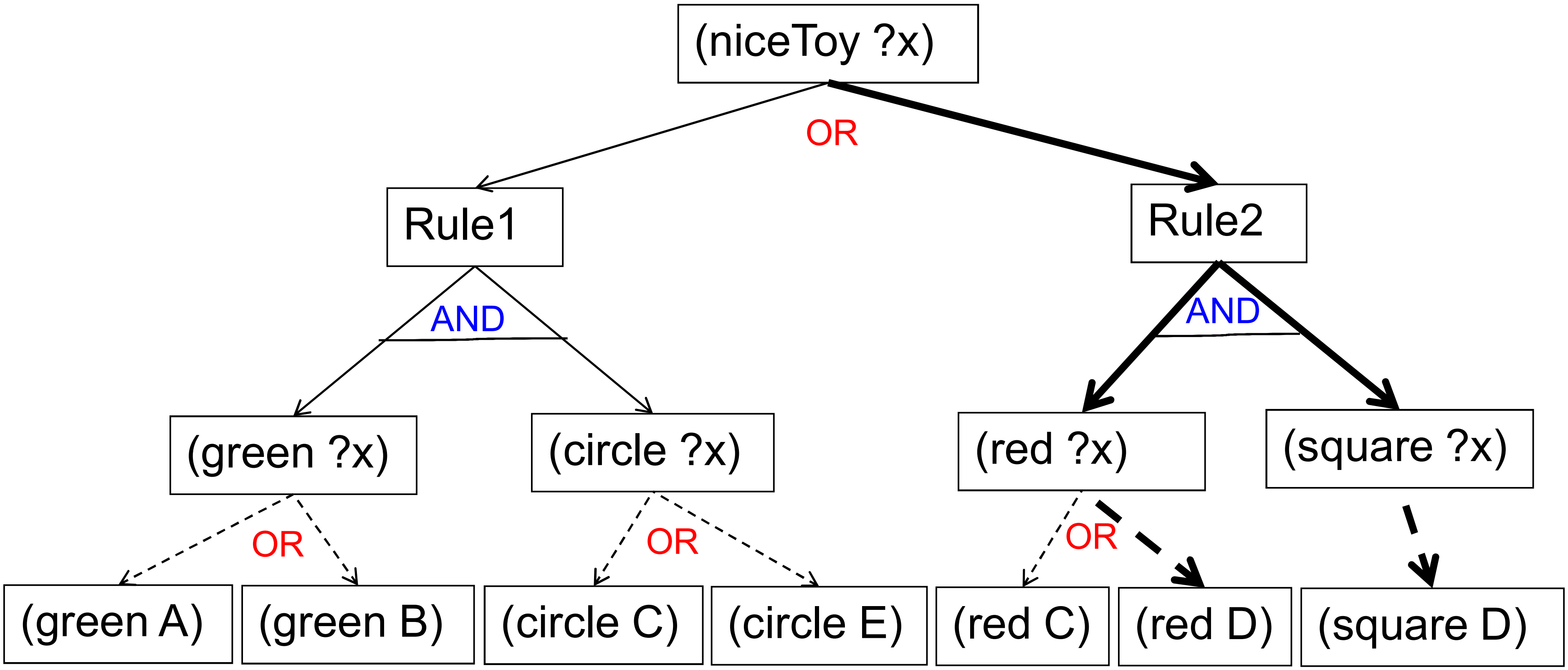
Depth First Search



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

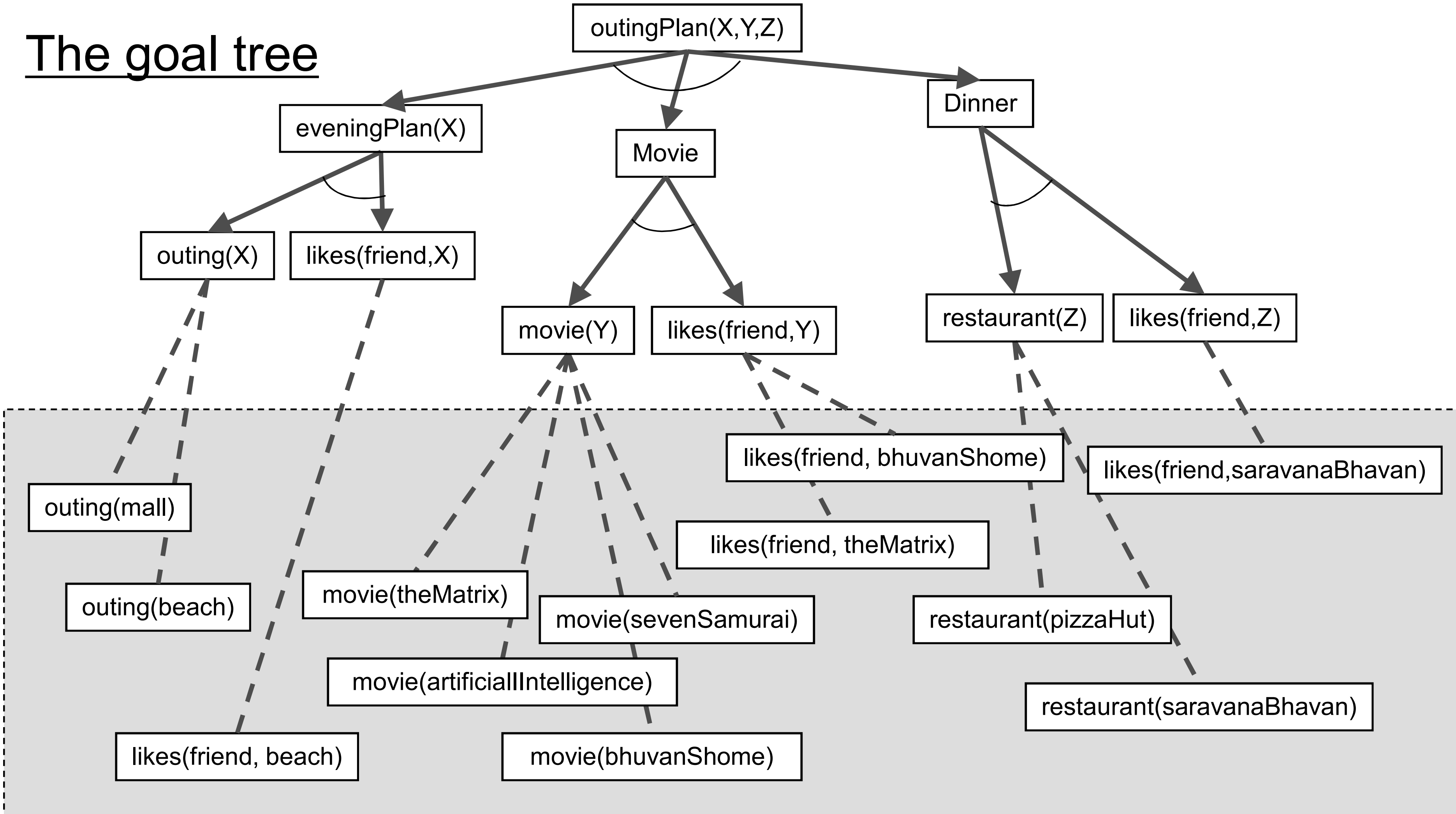


A Prolog KB (program)

```
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))
```

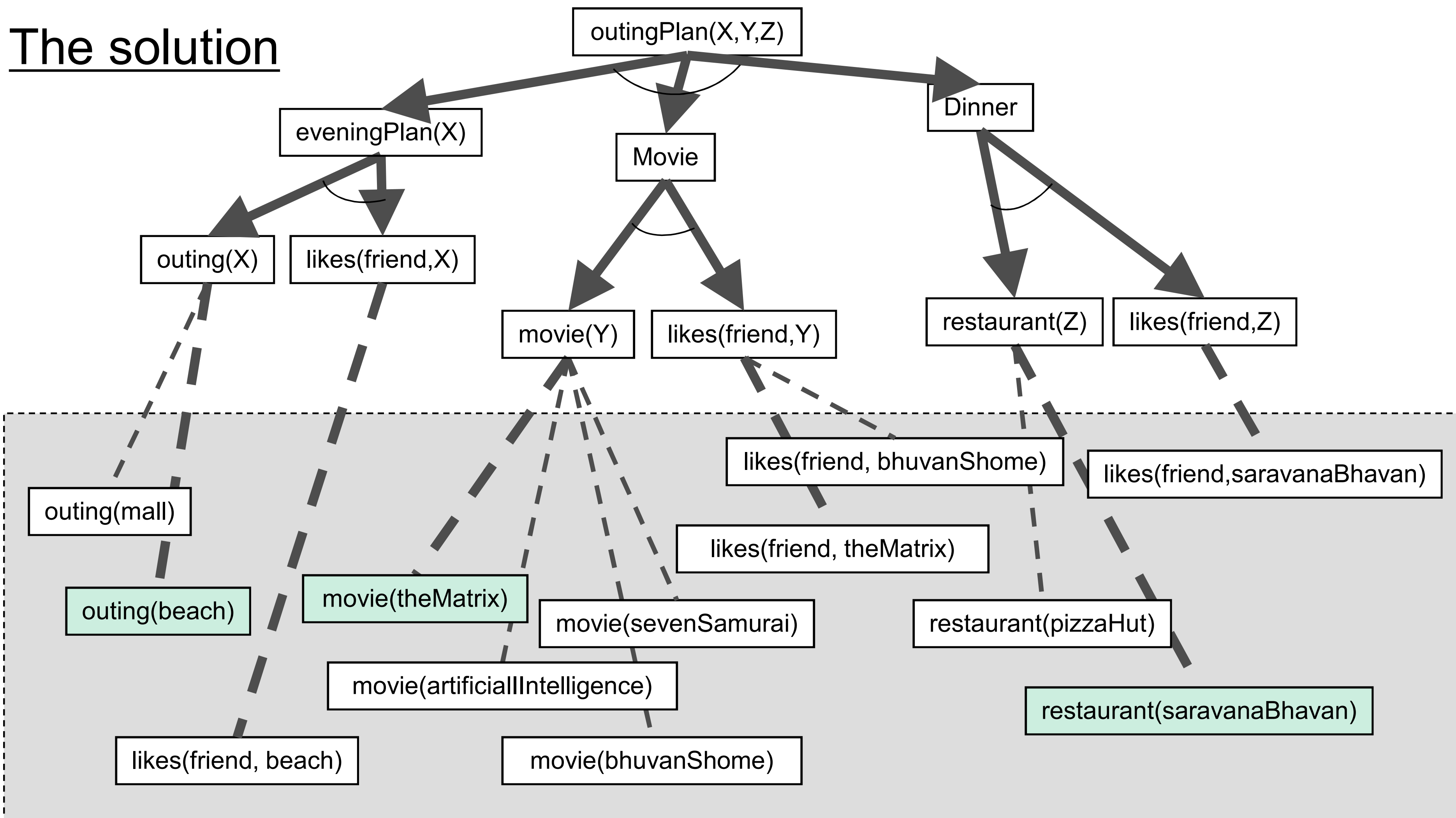
The goal tree



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

Given the following knowledge base (in list notation)

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

What is the KB talking about? What is the semantics?

Depends upon the interpretation $\vartheta = \langle D, I \rangle$!

Two sample interpretations….

Interpretation 1

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

Domain: Blocks World

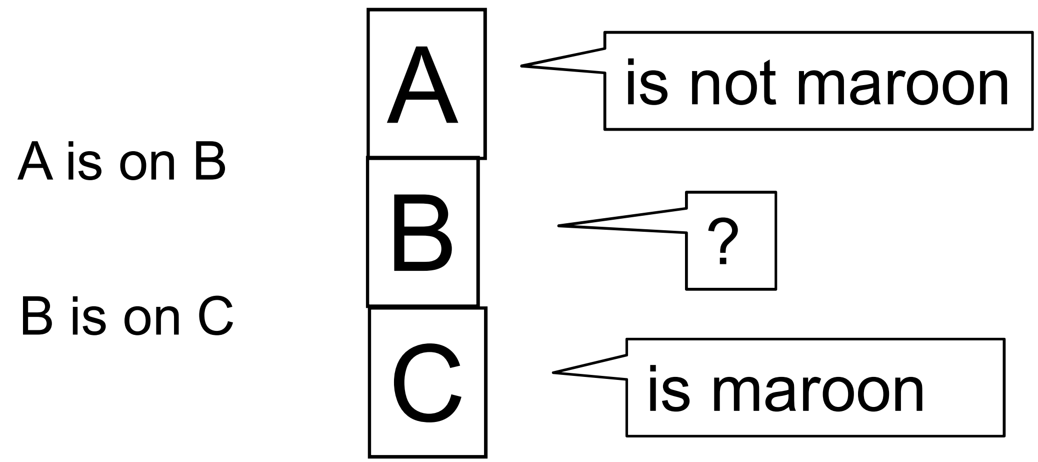
Predicate symbols

I(O) = On

I(M) = Maroon

Constant Symbols

A, B, C → blocks



Interpretation 2

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

Domain: People

Predicate symbols
I(O) = LookingAt
I(M) = Married

Constant Symbols
I(A) = Jack
I(B) = Anne
I(C) = John

Anne is looking at John

Jack is looking at Anne

John — is married

Anne — ?

Jack — is not married

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 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

End of Module 2