

Unit -2

Artificial Intelligence

NPTEL Links (Unit-2)

S.No	Topics (Associated with Unit - 2)	NPTEL Video Link:
1	Knowledge Representation and Logic	https://nptel.ac.in/courses/106/105/106105077/
2	Interface in Propositional Logic	
3	First Order Logic	
4	Reasoning Using First Order Logic	
5	Rule Based System	

NPTEL Links (Unit-2)

S.No	Topics (Associated with Unit -2)	NPTEL Video Link:
6	Rule Based Systems II	https://nptel.ac.in/courses/106/105/106105077/
7	Semantic Net	
8	Reasoning in Semantic Net	
9	Frames	

Knowledge Representation

- *Knowledge representation (KR)* is an important issue in both cognitive science and artificial intelligence.
 - In cognitive science, it is concerned with the way people store and process information and
 - In artificial intelligence (AI), main focus is to store knowledge so that programs can process it and achieve human intelligence.
- There are different ways of representing knowledge e.g.
 - predicate logic,
 - semantic networks,
 - extended semantic net,
 - frames,
 - conceptual dependency etc.
- In predicate logic, knowledge is represented in the form of rules and facts as is done in Prolog.

Semantic Network

- Formalism for representing information about objects, people, concepts and specific relationship between them.
- The syntax of semantic net is simple. It is a network of labeled nodes and links.
 - It's a directed graph with nodes corresponding to concepts, facts, objects etc. and
 - arcs showing relation or association between two concepts.
- The commonly used links in semantic net are of the following types.
 - **isa** → subclass of entity (e.g., child hospital is subclass of hospital)
 - **inst** → particular instance of a class (e.g., India is an instance of country)
 - **prop** → property link (e.g., property of dog is 'bark')

Representation of Knowledge in Sem Net

“Every human, animal and bird is living thing who breathe and eat. All birds can fly. All man and woman are humans who have two legs. Cat is an animal and has a fur. All animals have skin and can move. Giraffe is an animal who is tall and has long legs. Parrot is a bird and is green in color”.

Representation in Predicate Logic

- Every human, animal and bird is living thing who breathe and eat.

$\forall X$ [human(X) \rightarrow living(X)]

$\forall X$ [animal(X) \rightarrow living(X)]

$\forall X$ [bird(X) \rightarrow living(X)]

- All birds are animal and can fly.

$\forall X$ [bird(X) \wedge canfly(X)]

- Every man and woman are humans who have two legs.

$\forall X$ [man(X) \wedge haslegs(X)]

$\forall X$ [woman(X) \wedge haslegs(X)]

$\forall X$ [human(X) \wedge has(X, legs)]

- Cat is an animal and has a fur.

animal(cat) \wedge has(cat, fur)

- All animals have skin and can move.

$\forall X$ [animal(X) \rightarrow has(X, skin) \wedge canmove(X)]

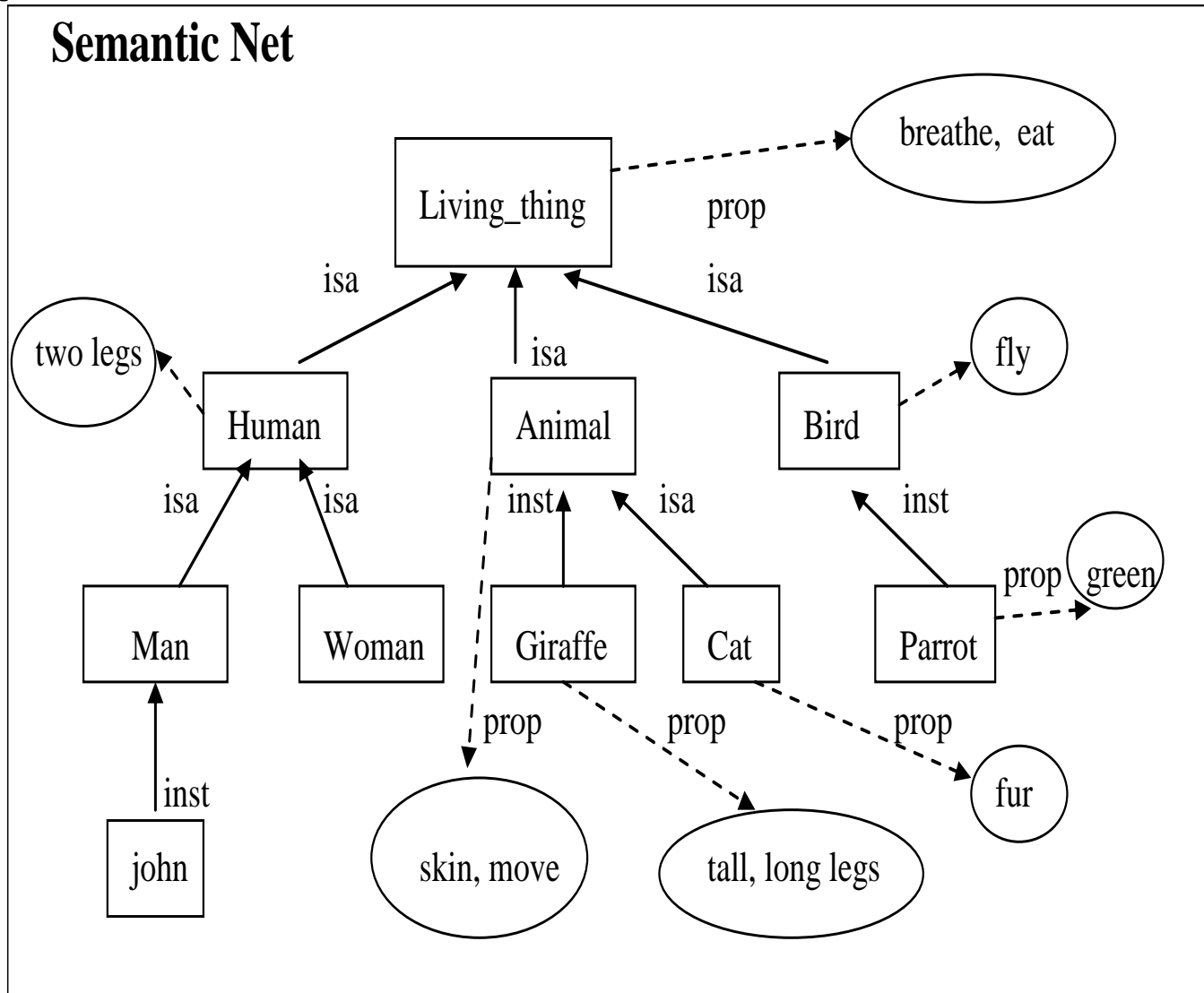
- Giraffe is an animal who is tall and has long legs.

animal(giraffe) \wedge has(giraffe, long_legs) \wedge is(giraffe, tall)

- Parrot is a bird and is green in color.

bird(parrot) \wedge has(parrot, green_colour)

Representation in Semantic Net



Inheritance

- Inheritance mechanism allows knowledge to be stored at the highest possible level of abstraction which reduces the size of knowledge base.
 - It facilitates inferencing of information associated with semantic nets.
 - It is a natural tool for representing taxonomically structured information and ensures that all the members and sub-concepts of a concept share common properties.
 - It also helps us to maintain the consistency of the knowledge base by adding new concepts and members of existing ones.
- Properties attached to a particular object (class) are to be inherited by all subclasses and members of that class.

Property Inheritance Algorithm

Input: Object, and property to be found from Semantic Net;

Output: Yes, if the object has the desired property else return false;

Procedure:

- Find an object in the semantic net; Found = false;
- While {(object \neq root) OR Found } DO
 - { If there is a a property attribute attached with an object then
 - { Found = true; Report 'Yes' } else
 - object=inst(object, class) OR isa(object, class)
- };
- If Found = False then report 'No'; Stop

Coding of Semantic Net in Prolog

Isa facts	Instance facts	Property facts
isa(living_thing, nil). isa(human, living_thing). isa(animals, living_thing). isa(birds, living_thing). isa(man, human). isa(woman, human). isa(cat, animal).	inst(john, man). inst(giraffe, animal). inst(parrot, bird)	prop(breathe, living_thing). prop(eat, living_thing). prop(two_legs, human). prop(skin, animal). prop(move, animal). prop(fur, bird). prop(tall, giraffe). prop(long_legs, giraffe). prop(tall, animal). prop(green, parrot).

Inheritance Rules in Prolog

Instance rules:

instance(X, Y) :- inst(X, Y).

instance(X, Y) :- inst(X, Z), subclass(Z, Y).

Subclass rules:

subclass(X, Y) :- isa(X, Y).

subclass(X, Y) :- isa(X, Z), subclass(Z, Y) .

Property rules:

property(X, Y) :- prop(X, Y).

property(X, Y) :- instance(Y, Z),
property(X, Z).

property(X, Y) :- subclass(Y, Z),
property(X, Z).

Queries

<ul style="list-style-type: none">● Is john human?● Is parrot a living thing?● Is giraffe an animal?● Is woman subclass of living thing● Does parrot fly?● Does john breathe?● has parrot fur?● Does cat fly?	<p>?- instance(john, humans). Y</p> <p>?- instance (parrot, living_thing). Y</p> <p>?- instance (giraffe, animal).Y</p> <p>?- subclass(woman, living_things). Y</p> <p>?- property(fly, parrot). Y</p> <p>?- property (john, breathe). Y</p> <p>?- property(fur, parrot). N</p> <p>?- property(fly, cat). N</p>
--	---

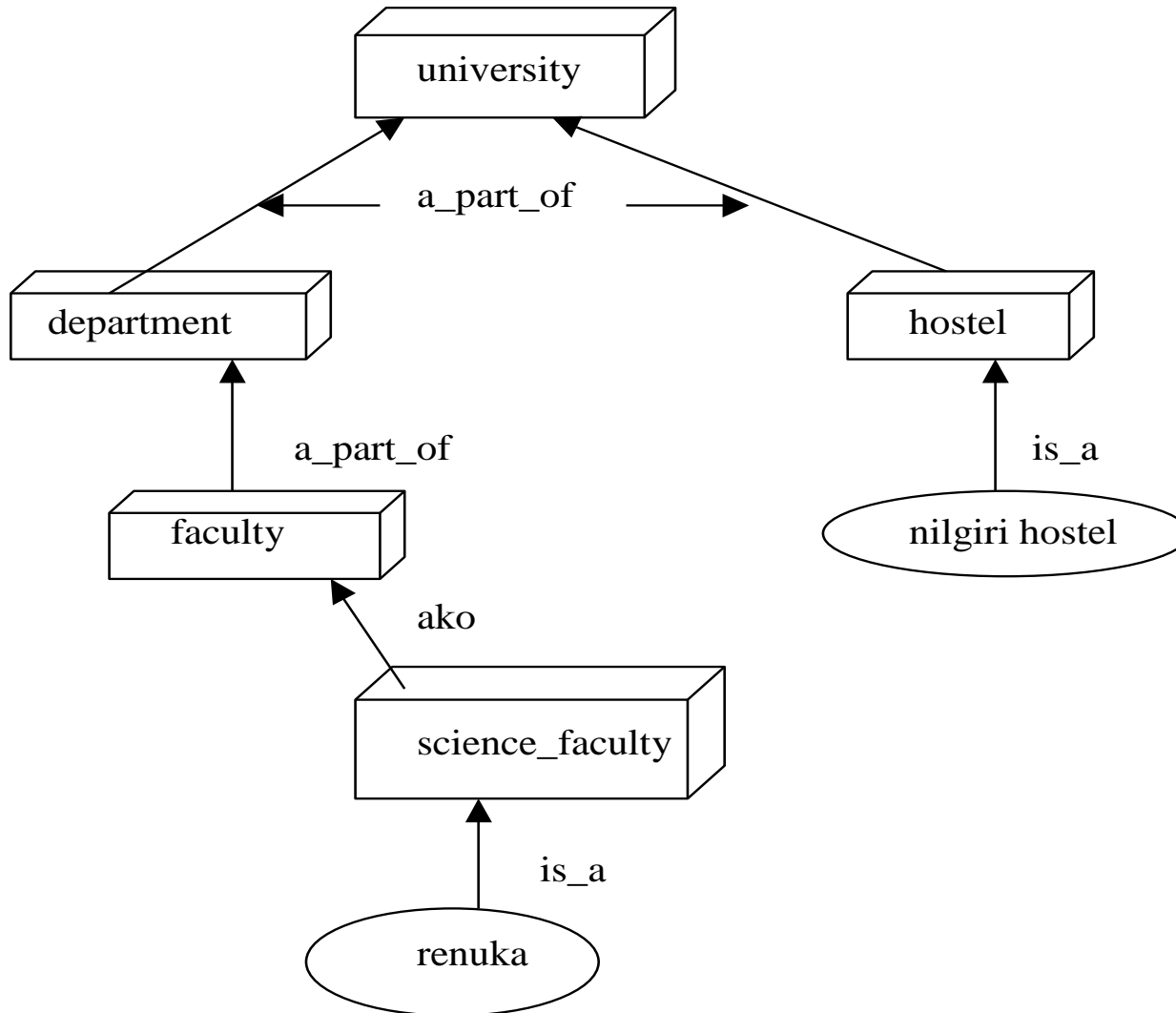
Knowledge Representation using Frames

- Frames are more structured form of packaging knowledge,
 - used for representing objects, concepts etc.
- Frames are organized into hierarchies or network of frames.
- Lower level frames can inherit information from upper level frames in network.
- Nodes are connected using links viz.,
 - **ako / subc** (links two class frames, one of which is subclass of other e.g., science_faculty class is **ako** of faculty class),
 - **is_a / inst** (connects a particular instance of a class frame e.g., Renuka **is_a** science_faculty)
 - **a_part_of** (connects two class frames one of which is contained in other e.g., faculty class **is_part_of** department class).
 - Property link of semantic net is replaced by SLOT fields.

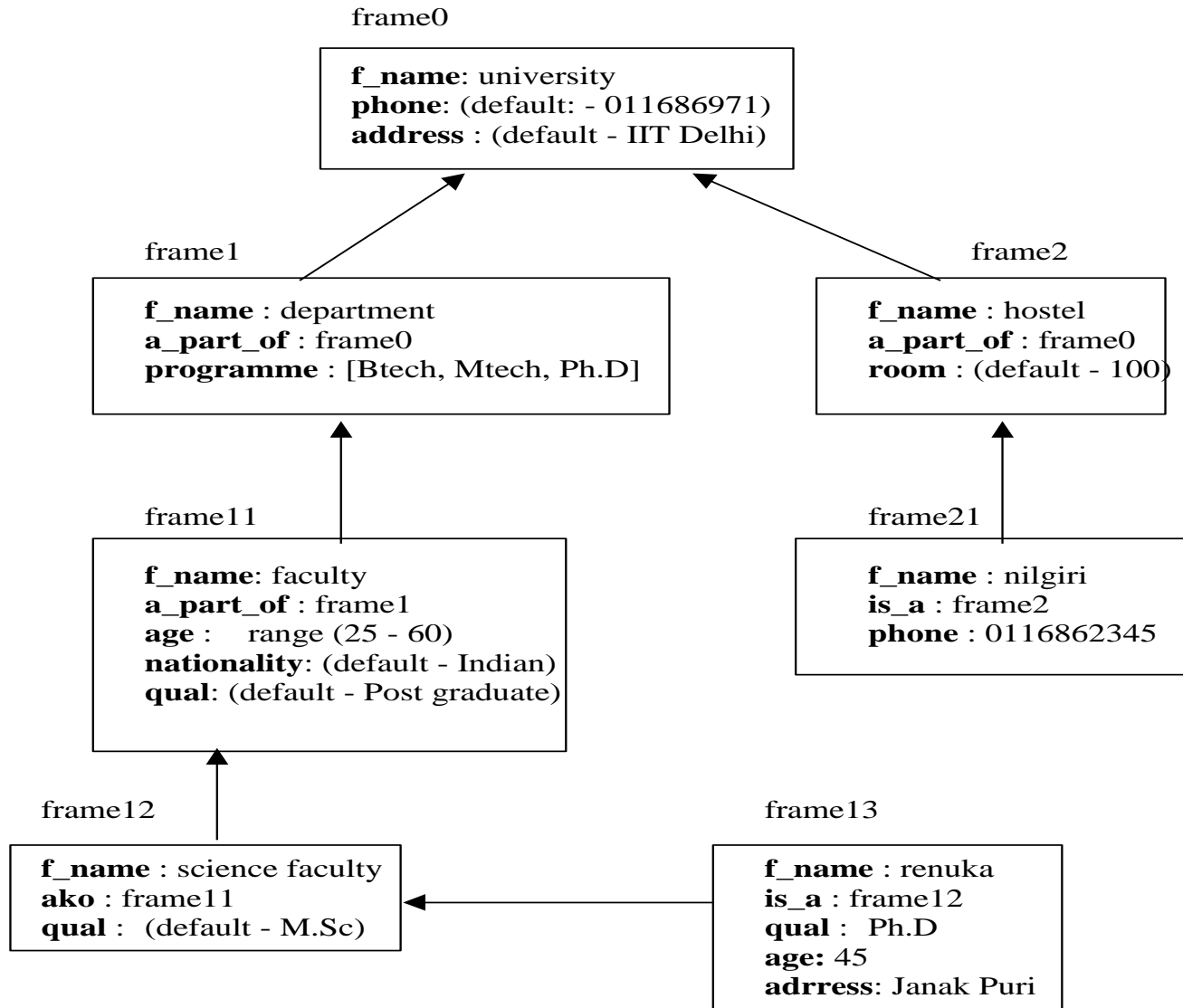
Cont...

- A frame may have any number of slots needed for describing object. e.g.,
 - faculty frame may have name, age, address, qualification etc as slot names.
- Each frame includes two basic elements : slots and facets.
 - Each slot may contain one or more **facets** (called fillers) which may take many forms such as:
 - **value** (value of the slot),
 - **default** (default value of the slot),
 - **range** (indicates the range of integer or enumerated values, a slot can have),
 - **demons** (procedural attachments such as if_needed, if_deleted, if_added etc.) and
 - **other** (may contain rules, other frames, semantic net or any type of other information).

Frame Network - Example



Detailed Representation of Frame Network



Description of Frames

- Each frame represents either a class or an instance.
- Class frame represents a general concept whereas instance frame represents a specific occurrence of the class instance.
- Class frame generally have default values which can be redefined at lower levels.
- If class frame has actual value facet then decedent frames can not modify that value.
- Value remains unchanged for subclasses and instances.

Inheritance in Frames

- Suppose we want to know nationality or phone of an instance-frame **frame13** of renuka.
- These informations are not given in this frame.
- Search will start from **frame13** in upward direction till we get our answer or have reached root frame.
- The frames can be easily represented in prolog by choosing predicate name as frame with two arguments.
- First argument is the name of the frame and second argument is a list of slot - facet pair.

Coding of frames in Prolog

```
frame(university, [phone (default, 011686971),  
                    address (default, IIT Delhi)]).
```

```
frame(deaprtment, [a_part_of (university),  
                  programme ([Btech, Mtech, Ph.d])]).
```

```
frame(hostel, [a_part_of (university), room(default, 100)]).
```

```
frame(faculty, [a_part_of (department), age(range,25,60),  
               nationality(default, indian), qual(default, postgraduate)]).
```

```
frame(nilgiri, [is_a (hostel), phone(011686234)]).
```

```
frame(science_faculty, [ako (faculty), qual(default, M.Sc.)]).
```

```
frame(renuka, [is_a (science_faculty), qual(Ph.D.), age(45),  
              address(janakpuri)]).
```

Inheritance Program in Prolog

```
find(X, Y) :- frame(X, Z), search(Z, Y), !.
```



```
find(X, Y) :- frame(X, [is_a(Z), _]), find(Z, Y), !.
```

```
find(X, Y) :- frame(X, [ako(Z), _]), find(Z, Y), !.
```

```
find(X, Y) :- frame(X, [a_part_of(Z), _]), find(Z, Y).
```

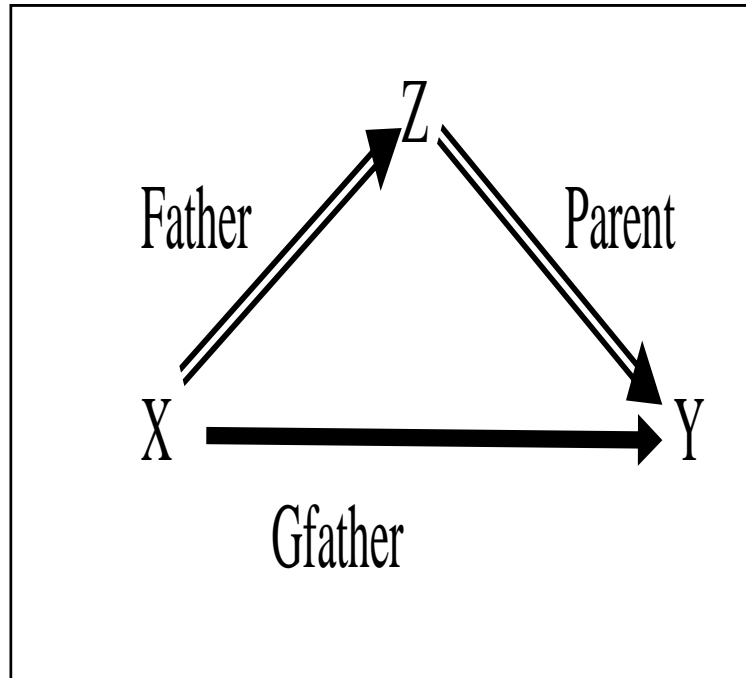
- Predicate **search** will basically retrieve the list of slots-facet pair and will try to match Y for slot.
- If match is found then its facet value is retrieved otherwise process is continued till we reach to root frame

Extended Semantic Network

- In conventional Sem Net, clausal form of logic can not be expressed.
- Extended Semantic Network (ESNet) combines the advantages of both logic and semantic network.
- In the ESNet, terms are represented by nodes similar to Sem Net.
- Binary predicate symbols in clausal logic are represented by labels on arcs of ESNet.
 - An *atom* of the form “Love(john, mary)” is an arc labeled as ‘Love’ with its two end nodes representing ‘john’ and ‘mary’.
- *Conclusions* and *conditions* in clausal form are represented by different kinds of arcs.
 - Conditions are drawn with two lines  and conclusions are drawn with one heavy line  .

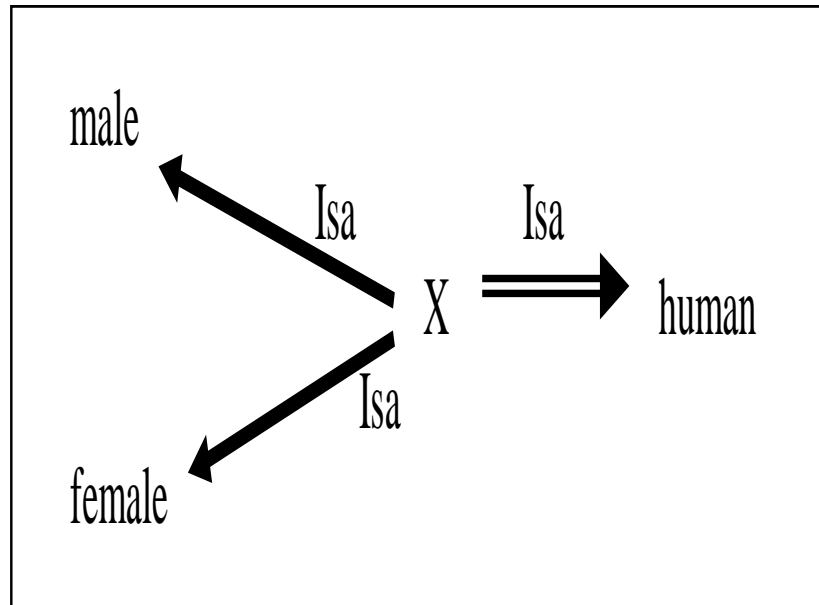
Examples

- Represent 'grandfather' definition
 $Gfather(X, Y) \leftarrow Father(X, Z), Parent(Z, Y)$ in
ESNet.



Cont...Example

- Represent clausal rule “**Male(X), Female(X) ← Human(X)**” using binary representation as “**Isa(X, male), Isa(X, female) ← Isa(X, human)**” and subsequently in ESNets as follows:

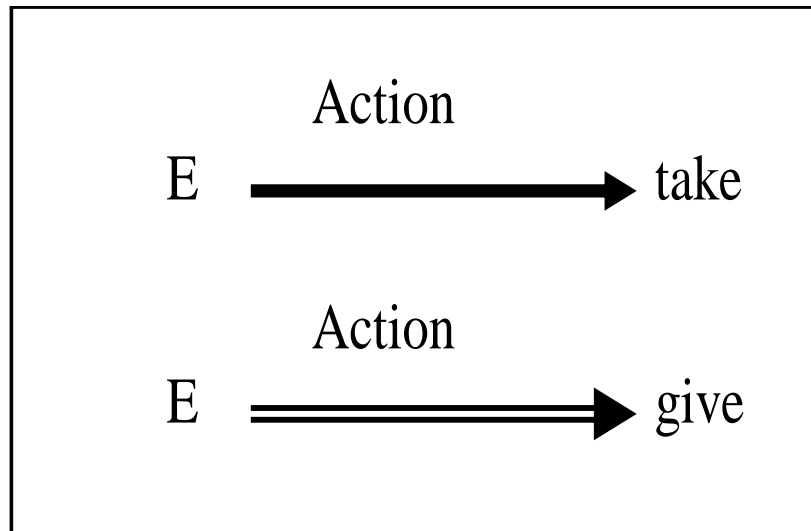


Inference Rules in ESNets

- Inference rules are embedded in the representation itself.
- The inference that “for every action of giving, there is an action of taking” in clausal logic written as

“Action(E, take) ← Action(E, give)”.

ESNet

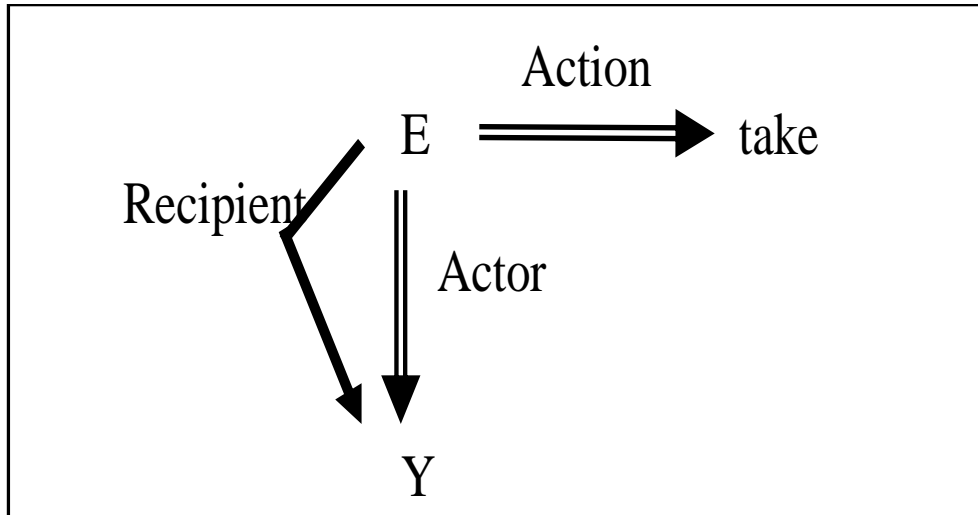


Cont...

- The inference rule such as “an actor of taking action is also the recipient of the action” can be easily represented in clausal logic as:
 - Here E is a variable representing an event where an action of taking is happening).

Recipient(E, Y) ← Acton(E, take), Actor (E, Y)

ESNet



Example

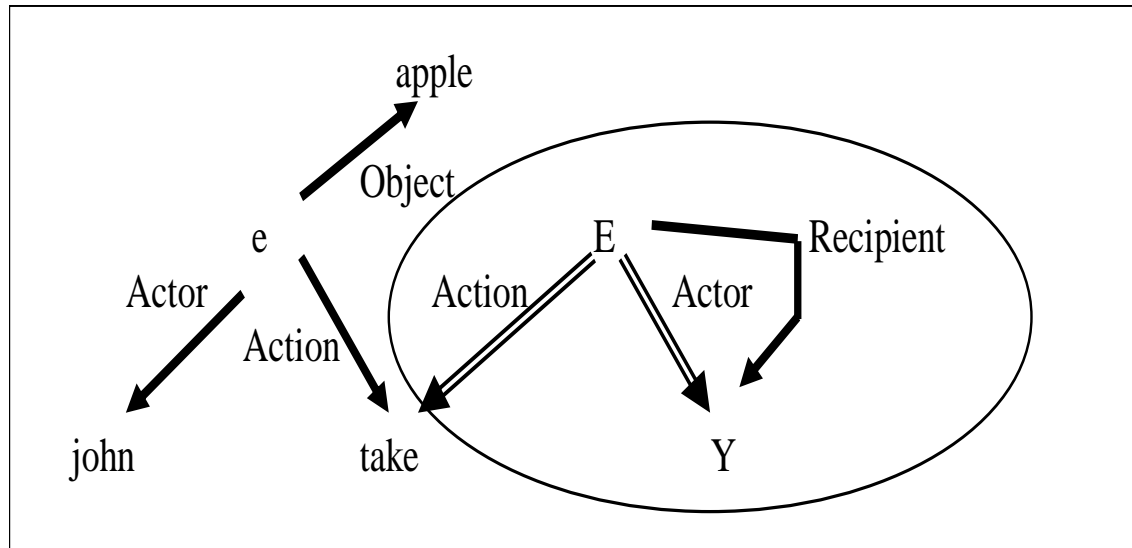
Represent the following clauses of Logic in ESNet.

Recipient(E, Y) \leftarrow Acton(E, take), Actor (E, Y)

Object (e, apple).

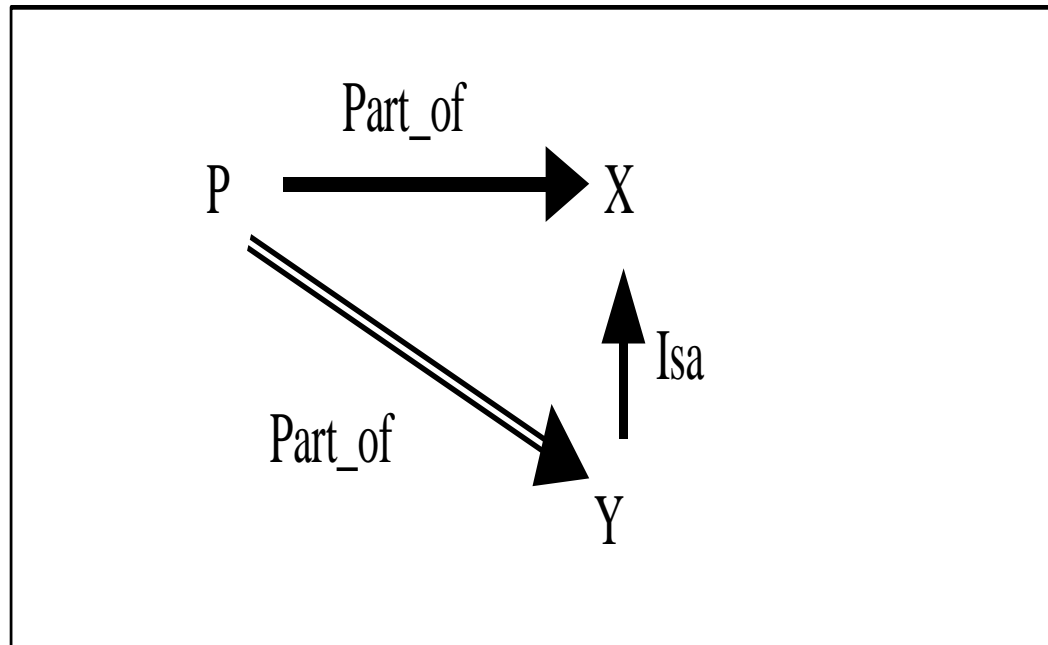
Action(e, take).

Actor (e, john) .



Contradiction

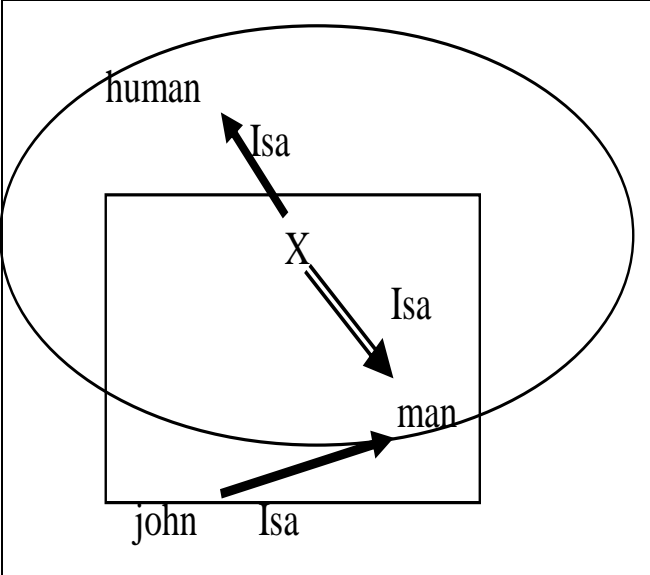
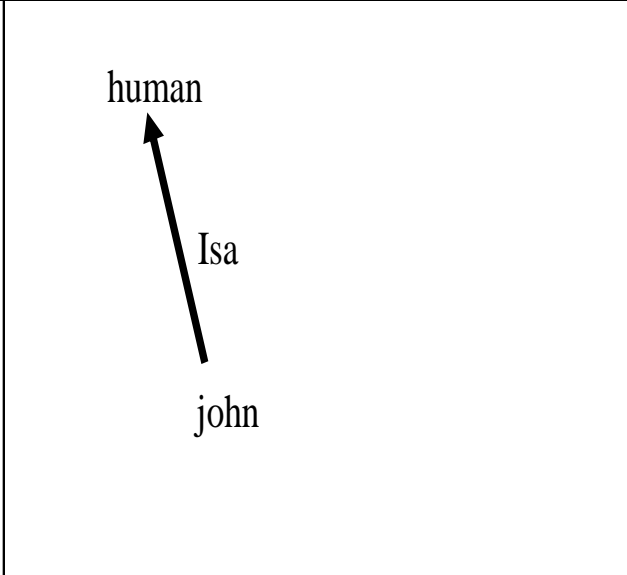
- The contradiction in the ESNets arises if we have the following situation.



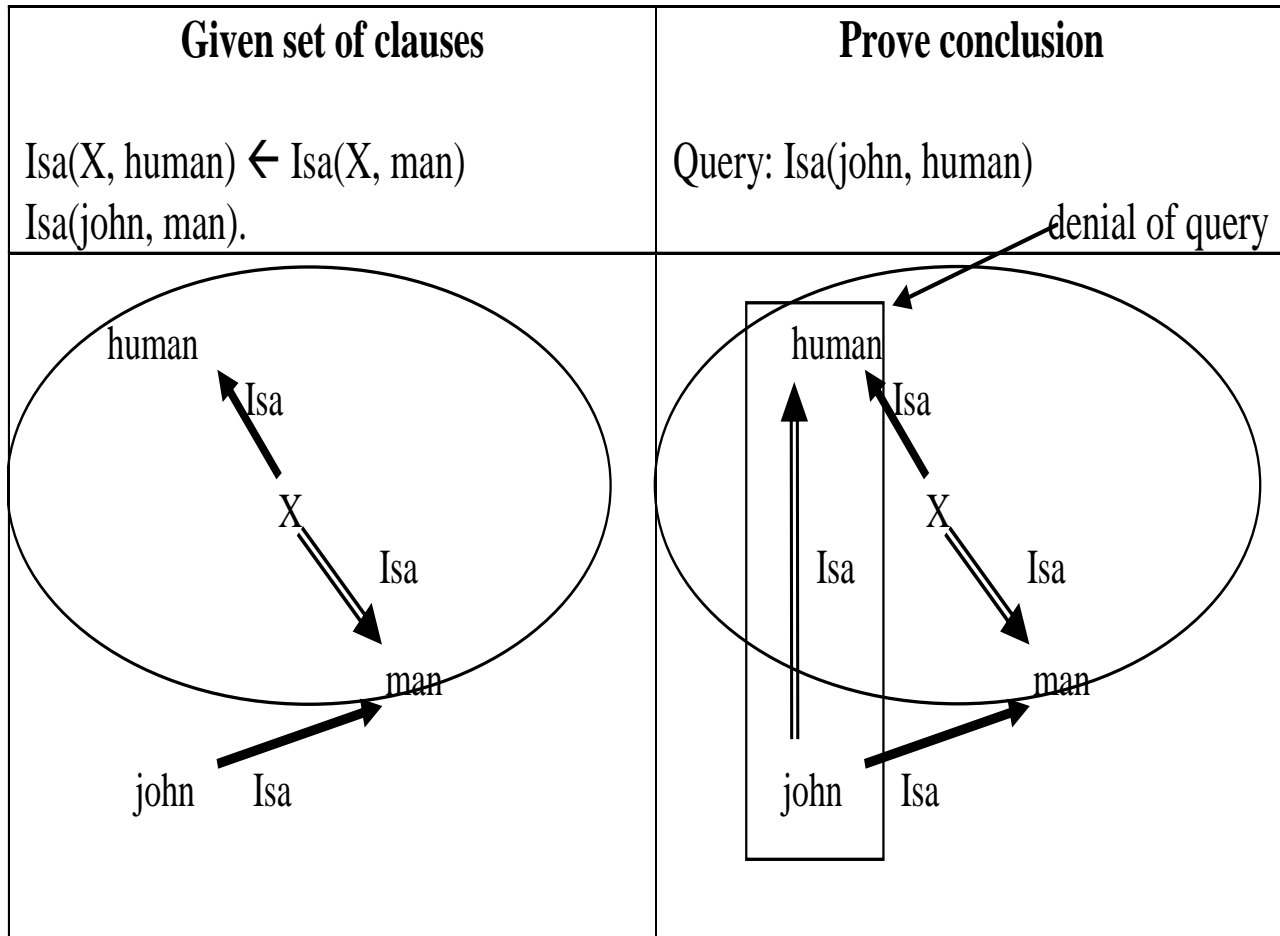
Deduction in ESNet

- Both of the following inference mechanisms are available in ESNet.
 - Forward reasoning inference (uses bottom up approach)
 - ***Bottom Up Inferencing:*** Given an ESNet, apply the following reduction (resolution) using modus ponens rule of logic ($\{A \leftarrow B, B\}$ then A).
 - Backward reasoning inference (uses top down approach).
 - ***Top Down Inferencing:*** Prove a conclusion from a given ESNet by adding the denial of the conclusion to the network and show that the resulting set of clauses in the network is inconsistent.

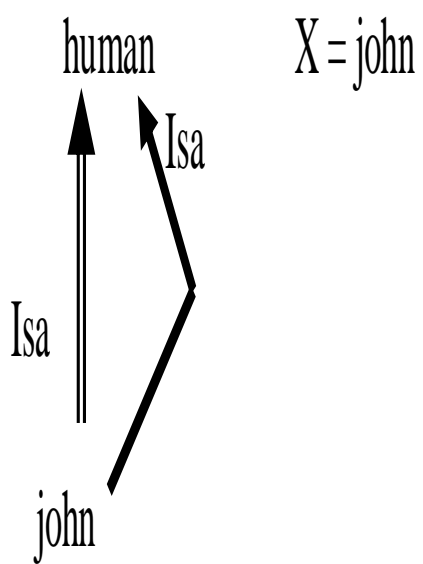
Example: Bottom Up Inferencing

Given set of clauses	Inferencing
<p>$\text{Isa}(X, \text{human}) \leftarrow \text{Isa}(X, \text{man})$ $\text{Isa}(\text{john}, \text{man}).$</p>	<p>$\text{Isa}(\text{john}, \text{human})$</p>
 <p>Here X is bound to john</p>	

Example: Top Down Inferencing



Cont...

 <p>human</p> <p>Isa</p> <p>Isa</p> <p>john</p> <p>X = john</p>	<p>Contradiction or Empty network is generated. Hence “Isa(john, human)” is proved.</p>
---	---

Logic and Deduction

- Logic is used to **formalize** deduction
- **Deduction** = **derivation** of true statements (called **conclusions**) from statements that are assumed to be true (called **premises**)
- Natural language is **not** precise, so the careless use of logic can lead to claims that false statements are true, or to claims that a statement is true, even though its truth does not necessarily follow from the premises
 - => Logic provides a way to talk about truth and correctness in a rigorous way, so that we can prove things, rather than make intelligent guesses and just hope they are correct

Why Propositional Logic?

- Propositional logic is a good vehicle to introduce basic properties of logic; used to:
 - **Associate** natural language expressions with semantic representations
 - **Evaluate** the truth or falsity of semantic representations relative to a knowledge base
 - **Compute** inferences over semantic representations
- One of the simplest and most common logic
 - The core of (almost) all other logics

What is Propositional Logic?

- An unambiguous formal language, akin to a programming language
 - **Syntax**: Vocabulary for expressing concepts without ambiguity
 - **Semantics**: Connection to what we're reasoning about
 - *Interpretation* - what the syntax means
 - **Reasoning**: How to prove things
 - What steps are allowed

TECHNICAL SOLUTIONS

SYNTAX

Syntax

- **Logical constants:** true, false
- **Propositional symbols:** P, Q, S, ...
- **Wrapping parentheses:** (...)
- **Atomic formulas:** Propositional Symbols or logical constants
- **Formulas** are either atomic formulas, or can be formed by combining atomic formulas with the following **connectives**:

\wedge ...and [conjunction]

\vee ...or [disjunction]

\rightarrow ...implies [implication / conditional]

\leftrightarrow ..is equivalent [biconditional]

\neg ...not [negation]

Syntax (cont')

- A **sentence** (well formed formula) is defined as follows:
 - A symbol is a sentence
 - If S is a sentence, then $\neg S$ is a sentence
 - If S is a sentence, then (S) is a sentence
 - If S and T are sentences, then $(S \vee T)$, $(S \wedge T)$, $(S \rightarrow T)$, and $(S \leftrightarrow T)$ are sentences
 - A sentence results from a finite number of applications of the above rules

Syntax – BNF Grammar

Sentence → *AtomicSentence* | *ComplexSentence*

AtomicSentence → True | False | P | Q | R | ...

ComplexSentence → (*Sentence*)
| *Sentence* *Connective* *Sentence*
| ¬ *Sentence*

Connective → ∧ | ∨ | → | ↔

Ambiguities are resolved through precedence ¬ ∧ ∨ → ↔
or parentheses

e.g. ¬ P ∨ Q ∧ R ⇒ S is equivalent to (¬ P) ∨ (Q ∧ R) ⇒ S

Syntax – Examples

- P means “It is hot.”
- Q means “It is humid.”
- R means “It is raining.”
- $(P \wedge Q) \rightarrow R$
“If it is hot and humid, then it is raining”
- $Q \rightarrow P$
“If it is humid, then it is hot”
- $\neg p \wedge \neg q$
- $\neg(p \wedge q)$
- $(p \wedge q) \vee r$
- $p \wedge q \wedge r$
- $((\neg p) \wedge q) \rightarrow r$
 $\leftrightarrow ((\neg r) \vee p)$
- $(\neg (p \vee q) \rightarrow q) \rightarrow r$
- $((\neg p) \vee (\neg q)) \leftrightarrow (\neg r)$
- Etc.

SEMANTICS

Semantics

- Interpretations
- Equivalence
- Substitution
- Models and Satisfiability
- Validity
- Logical Consequence (Entailment)
- Theory

Semantics – Some Informal Definitions

- Given the truth values of all symbols in a sentence, it can be “evaluated” to determine its **truth value** (True or False)
- A **model** for a KB is a “possible world” (assignment of truth values to propositional symbols) in which each sentence in the KB is True
- A **valid sentence** or **tautology** is a sentence that is True under all interpretations, no matter what the world is actually like or how the semantics are defined (example: “It’s raining or it’s not raining”)
- An **inconsistent sentence** or **contradiction** is a sentence that is False under all interpretations (the world is never like what it describes, as in “It’s raining and it’s not raining”)
- **P entails Q**, written $P \models Q$, means that whenever P is True, so is Q; in other words, all models of P are also models of Q

Interpretations

- In propositional logic, truth values are assigned to the atoms of a formula in order to evaluate the truth value of the formula
- An **assignment** is a function

$$v : P \rightarrow \{T, F\}$$

v assigns a **truth value** to any atom in a given formula (P is the set of all propositional letters, i.e. **atoms**)

Suppose F denotes the set of all propositional formulas. We can extend an assignment v to a function

$$v : F \rightarrow \{T, F\}$$

which assigns the truth value $v(A)$ to any formula A in F . v is called an **interpretation**.

Interpretations (cont')

- Example:

- Suppose v is an assignment for which

$$v(p) = F, v(q) = T.$$

- If $A = (\neg p \rightarrow q) \leftrightarrow (p \vee q)$, what is $v(A)$?

Solution:

$$\begin{aligned}v(A) &= v((\neg p \rightarrow q) \leftrightarrow (p \vee q)) \\&= v(\neg p \rightarrow q) \leftrightarrow v(p \vee q) \\&= (v(\neg p) \rightarrow v(q)) \leftrightarrow (v(p) \vee v(q)) \\&= (\neg v(p) \rightarrow v(q)) \leftrightarrow (v(p) \vee v(q)) \\&= (\neg F \rightarrow T) \leftrightarrow (F \vee T) \\&= (T \rightarrow T) \leftrightarrow (F \vee T) \\&= T \leftrightarrow T \\&= T\end{aligned}$$

Equivalence

- If A, B are formulas are such that

$$v(A) = v(B)$$

for **all** interpretations v , A is **(logically) equivalent** to B :

$$A \equiv B$$

- Example: $\neg p \vee q \equiv p \rightarrow q$ since both formulas are true in all interpretations except when $v(p) = T$, $v(q) = F$ and are false for that particular interpretation
 - Caution: \equiv does **not** mean the same thing as \leftrightarrow :
 - $A \leftrightarrow B$ is a **formula** (syntax)
 - $A \equiv B$ is a **relation** between two formula (semantics)
- Theorem: $A \equiv B$ if and only if $A \leftrightarrow B$ is true in every interpretation; i.e. $A \leftrightarrow B$ is a **tautology**.

Equivalence and Substitution – Examples

- Examples of logically equivalent formulas

$$A \equiv \neg\neg A$$

$$A \vee B \equiv B \vee A$$

$$(A \vee B) \vee C \equiv A \vee (B \vee C)$$

$$A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$$

$$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$$

$$\neg(A \wedge B) \equiv \neg A \vee \neg B$$

$$A \wedge \text{true} \equiv A$$

$$A \wedge B \equiv B \wedge A$$

$$(A \wedge B) \wedge C \equiv A \wedge (B \wedge C)$$

$$\neg(A \vee B) \equiv \neg A \wedge \neg B$$

$$A \rightarrow \text{false} \equiv \neg A$$

- Example: Simplify $p \vee (\neg p \wedge q)$

– Solution:

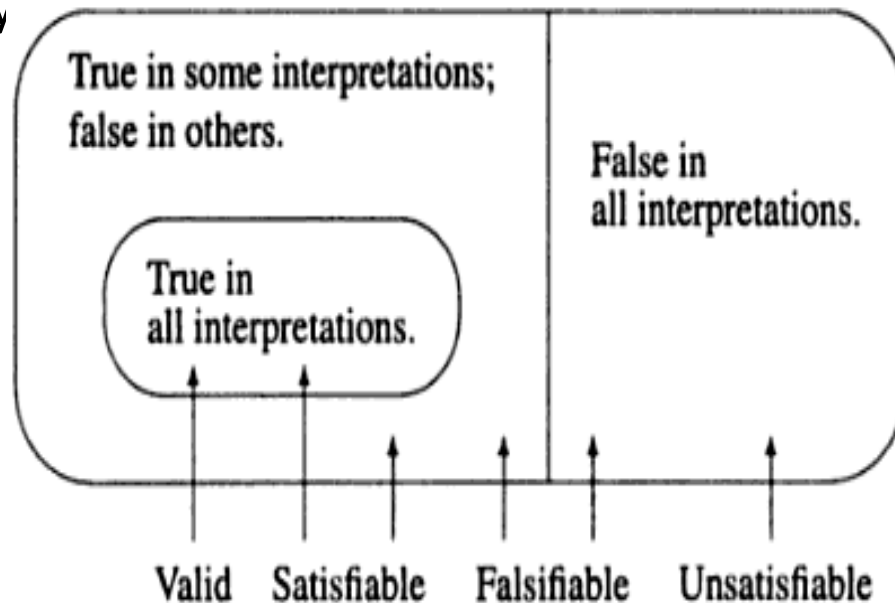
$$\begin{aligned} p \vee (\neg p \wedge q) &\equiv (p \vee \neg p) \wedge (p \vee q) \\ &\equiv \text{T} \wedge (p \vee q) \\ &\equiv p \vee q \end{aligned}$$

Models and Satisfiability

- A propositional **formula** A is **satisfiable** iff $v(A) = T$ in **some** interpretation v ; such an interpretation is called a **model** for A .
 - A is **unsatisfiable** (or, **contradictory**) if it is false in every interpretation
- A **set of formulas** $U = \{A_1, A_2, \dots, A_n\}$ is **satisfiable** iff there exists an interpretation v such that $v(A_1) = v(A_2) = \dots = v(A_n) = T$; such an interpretation is called a **model** of U .
 - U is **unsatisfiable** if no such interpretation exists
- Relevant properties:
 - If U is satisfiable, then so is $U - \{A_i\}$ for any $i = 1, 2, \dots, n$
 - If U is satisfiable and B is valid, then $U \cup \{B\}$ is also satisfiable
 - If U is unsatisfiable and B is **any** formula, $U \cup \{B\}$ is also unsatisfiable
 - If U is unsatisfiable and some A_i is valid, then $U - \{A_i\}$ is also unsatisfiable

Validity

- A is **valid** (or, a **tautology**), denoted $\models A$, iff $v(A) = T$, for **all** interpretations v
- A is **not valid** (or, **falsifiable**), denoted $\not\models A$ if we can find **some** interpretation v , such that $v(A) = F$
- **Relationship** between validity, satisfiability, falsifiability, and unsatisfiability



Validity (cont')

- Examples:

- Valid (tautology): $(p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p)$

- Not valid, but satisfiable: $q \rightarrow (q \rightarrow p)$

- False (contradiction): $(p \wedge \neg p) \vee (q \wedge \neg q)$

- Theorem:

(a) A is valid if and only if $\neg A$ is unsatisfiable

(b) A is satisfiable if and only if $\neg A$ is falsifiable

Logical Consequence (i.e. Entailment)

- Let U be a set of formulas and A a formula. A is a **(logical) consequence** of U , if any interpretation v which is a model of U is also a model for A :

$$U \models A$$

- Example: $\{p \wedge r, \neg q \vee (p \wedge \neg p)\} \models (p \wedge \neg q) \rightarrow r$

If some interpretation v is a model for the set

$\{p \wedge r, \neg q \vee (p \wedge \neg p)\}$ must satisfy

$$v(p) = v(r) = T, \quad v(q) = F$$

but in this interpretation, we also have

$$v((p \wedge \neg q) \rightarrow r) = T$$

Theory

- A set of formulas T is a **theory** if it is closed under logical consequence. This means that, for every formula A , if $T \vDash A$, then A is in T
- Let U be a set of formulas. Then, the set of all consequences of U

$$T(U) = \{A \mid U \vDash A\}$$

is called the **theory** of U .

The formulas in U are called the **axioms** for the theory $T(U)$.

INFERENCE

Inference Methods

- Several basic methods for determining whether a given set of premises propositionally entails a given conclusion
 - Truth Table Method
 - Deductive (Proof) Systems
 - Resolution

Truth Table Method

- One way of determining whether or not a set of premises logically entails a possible conclusion is to check the truth table for the logical constants of the language
- This is called the **truth table method** and can be formalized as follows:
 - **Step 1:** Starting with a complete truth table for the propositional constants, iterate through all the premises of the problem, for each premise eliminating any row that does not satisfy the premise
 - **Step 2:** Do the same for the conclusion
 - **Step 3:** Finally, compare the two tables; If every row that remains in the premise table, i.e. is not eliminated, also remains in the conclusion table, i.e. is not eliminated, then the premises logically entail the conclusion

Example

- Simple sentences:
 - Amy loves Pat: *lovesAmyPat*
 - Amy loves Quincy: *lovesAmyQuincy*
 - It is Monday: *ismonday*
- Premises:
 - If Amy loves Pat, Amy loves Quincy:
lovesAmyPat \rightarrow *lovesAmyQuincy*
 - If it is Monday, Amy loves Pat or Quincy:
ismonday \rightarrow *lovesAmyPat* \vee *lovesAmyQuincy*
- Question:
 - If it is Monday, does Amy love Quincy?
i.e. is *ismonday* \rightarrow *lovesAmyQuincy* entailed by the premises?

Step 1: Truth table for the premises

<i>lovesAmyP at</i>	<i>lovesAmyQu incy</i>	<i>ismonday</i>	<i>lovesAmyPa t → lovesAmyQu incy</i>	<i>ismonday → lovesAmyPa t ∨ lovesAmyQu incy</i>
<i>T</i>	<i>T</i>	<i>T</i>	<i>T</i>	<i>T</i>
<i>T</i>	<i>T</i>	<i>F</i>	<i>T</i>	<i>T</i>
<i>T</i>	<i>F</i>	<i>T</i>	<i>F</i>	<i>T</i>
<i>T</i>	<i>F</i>	<i>F</i>	<i>F</i>	<i>T</i>
<i>F</i>	<i>T</i>	<i>T</i>	<i>T</i>	<i>T</i>
<i>F</i>	<i>T</i>	<i>F</i>	<i>T</i>	<i>T</i>
<i>F</i>	<i>F</i>	<i>T</i>	<i>T</i>	<i>F</i>
<i>F</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>T</i>

Step 1: Eliminate non-sat interpretations

<i>lovesAmyP at</i>	<i>lovesAmyQu incy</i>	<i>ismonday</i>	<i>lovesAmyPa t → lovesAmyQu incy</i>	<i>ismonday → lovesAmyPa t ∨ lovesAmyQu incy</i>
<i>T</i>	<i>T</i>	<i>T</i>	T	T
<i>T</i>	<i>T</i>	<i>F</i>	T	T
<i>T</i>	<i>F</i>	<i>T</i>	F	T
<i>T</i>	<i>F</i>	<i>F</i>	F	T
<i>F</i>	<i>T</i>	<i>T</i>	T	T
<i>F</i>	<i>T</i>	<i>F</i>	T	T
<i>F</i>	<i>F</i>	<i>T</i>	T	F
<i>F</i>	<i>F</i>	<i>F</i>	T	T

Step 2: Truth table for the conclusion

<i>lovesAmyP at</i>	<i>lovesAmyQu incy</i>	<i>ismonday</i>	<i>ismonday → lovesAmyQuincy</i>
<i>T</i>	<i>T</i>	<i>T</i>	<i>T</i>
<i>T</i>	<i>T</i>	<i>F</i>	<i>T</i>
<i>T</i>	<i>F</i>	<i>T</i>	<i>F</i>
<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>
<i>F</i>	<i>T</i>	<i>T</i>	<i>T</i>
<i>F</i>	<i>T</i>	<i>F</i>	<i>T</i>
<i>F</i>	<i>F</i>	<i>T</i>	<i>F</i>
<i>F</i>	<i>F</i>	<i>F</i>	<i>T</i>

Step 2: Eliminate non-sat interpretations

<i>lovesAmyP at</i>	<i>lovesAmyQu incy</i>	<i>ismonday</i>	<i>ismonday → lovesAmyQuincy</i>
<i>T</i>	<i>T</i>	<i>T</i>	T
<i>T</i>	<i>T</i>	<i>F</i>	T
<i>T</i>	<i>F</i>	<i>T</i>	F
<i>T</i>	<i>F</i>	<i>F</i>	T
<i>F</i>	<i>T</i>	<i>T</i>	T
<i>F</i>	<i>T</i>	<i>F</i>	T
<i>F</i>	<i>F</i>	<i>T</i>	F
<i>F</i>	<i>F</i>	<i>F</i>	T

Step 3: Comparing tables

- Finally, in order to make the determination of logical entailment, we compare the two rightmost tables and notice that every row remaining in the premise table also remains in the conclusion table.
 - In other words, the premises logically entail the conclusion.
- The truth table method has the merit that it is easy to understand
 - It is a direct implementation of the definition of logical entailment.
- In practice, it is awkward to manage two tables, especially since there are simpler approaches in which *only one table* needs to be manipulated
 - **Validity Checking**
 - **Unsatisfiability Checking**

Validity checking

- Approach: To determine whether a set of sentences

$$\{\varphi_1, \dots, \varphi_n\}$$

logically entails a sentence φ , form the sentence

$$(\varphi_1 \wedge \dots \wedge \varphi_n \rightarrow \varphi)$$

and check that it is valid.

- To see how this method works, consider the previous example and write the tentative conclusion as shown below.

$$(\text{lovesAmyPat} \rightarrow \text{lovesAmyQuincy}) \wedge (\text{ismondays} \rightarrow \text{lovesAmyPat} \vee \text{lovesAmyQuincy}) \\ \rightarrow (\text{ismondays} \rightarrow \text{lovesAmyQuincy})$$

- Then, form a truth table for our language with an added column for this sentence and check its satisfaction under each of the possible interpretations for our logical constants

Unsatisfiability Checking

- It is almost exactly the same as the validity checking method, except that it works negatively instead of positively.
- To determine whether a finite set of sentences $\{\varphi_1, \dots, \varphi_n\}$ logically entails a sentence φ , we form the sentence
$$(\varphi_1 \wedge \dots \wedge \varphi_n \wedge \neg\varphi)$$
and check that it is *unsatisfiable*.
- Both the validity checking method and the satisfiability checking method require about the same amount of work as the truth table method, but they have the merit of manipulating **only one table**

Deductive (proof) systems

- Semantic methods for checking logical entailment have the merit of being conceptually simple; they directly manipulate interpretations of sentences
- Unfortunately, the number of interpretations of a language grows **exponentially** with the number of logical constants.
 - When the number of logical constants in a propositional language is large, the number of interpretations may be impossible to manipulate.
- **Deductive (proof) systems** provide an alternative way of checking and communicating logical entailment that addresses this problem
 - In many cases, it is possible to create a “proof” of a conclusion from a set of premises that is much smaller than the truth table for the language;
 - Moreover, it is often possible to find such proofs with less work than is necessary to check the entire truth table

Schemata

- An important component in the treatment of proofs is the notion of a **schema**
- A *schema* is an expression satisfying the grammatical rules of our language except for the occurrence of *metavariables* in place of various subparts of the expression.
 - For example, the following expression is a pattern with metavariables φ and ψ .

$$\varphi \rightarrow (\psi \rightarrow \varphi)$$

- An **instance** of a sentence schema is the expression obtained by substituting expressions for the metavariables.
 - For example, the following is an instance of the preceding schema.

$$p \rightarrow (q \rightarrow p)$$

Rules of Inference

- The basis for proof systems is the use of correct rules of inference that can be applied directly to sentences to derive conclusions that are guaranteed to be correct under all interpretations
 - Since the interpretations are not enumerated, time and space can often be saved
- A **rule of inference** is a pattern of reasoning consisting of:
 - One set of sentence schemata, called *premises*, and
 - A second set of sentence schemata, called *conclusions*
- A rule of inference is **sound** if and only if, for every instance, the premises logically entail the conclusions

E.g. Modus Ponens (MP)

$$\varphi \rightarrow \psi$$
$$\varphi$$

$$\psi$$
$$p \rightarrow (q \rightarrow r)$$
$$p$$

$$q \rightarrow r$$
$$\textit{raining} \rightarrow \textit{wet}$$
$$\textit{raining}$$

$$\textit{wet}$$
$$(p \rightarrow q) \rightarrow r$$
$$p \rightarrow q$$

$$r$$
$$\textit{wet} \rightarrow$$
$$\textit{slippery}$$

$$\textit{wet}$$
$$\textit{slippery}$$

- I.e. we can substitute for the metavariables complex sentences
- Note that, by stringing together applications of rules of inference, it is possible to derive conclusions that cannot be derived in a single step. This idea of stringing together rule applications leads to the notion of a proof.

Axiom schemata

- The *implication introduction schema (II)*, together with Modus Ponens, allows us to infer implications
$$\varphi \rightarrow (\psi \rightarrow \varphi)$$
- The *implication distribution schema (ID)* allows us to distribute one implication over another
$$(\varphi \rightarrow (\psi \rightarrow \chi)) \rightarrow ((\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \chi))$$
- The *contradiction realization schemata (CR)* permit us to infer a sentence if the negation of that sentence implies some sentence and its negation
$$(\psi \rightarrow \neg\varphi) \rightarrow ((\psi \rightarrow \varphi) \rightarrow \neg\psi)$$
$$(\neg\psi \rightarrow \neg\varphi) \rightarrow ((\neg\psi \rightarrow \varphi) \rightarrow \psi)$$

Axiom schemata (cont')

- The *equivalence schemata (EQ)* captures the meaning of the \leftrightarrow operator
$$(\varphi \leftrightarrow \psi) \rightarrow (\varphi \rightarrow \psi)$$
$$(\varphi \leftrightarrow \psi) \rightarrow (\psi \rightarrow \varphi)$$
$$(\varphi \rightarrow \psi) \rightarrow ((\psi \rightarrow \varphi) \rightarrow (\psi \leftrightarrow \varphi))$$
- The meaning of the other operators in propositional logic is captured in the following axiom schemata
$$(\varphi \leftarrow \psi) \leftrightarrow (\psi \rightarrow \varphi)$$
$$(\varphi \vee \psi) \leftrightarrow (\neg\varphi \rightarrow \psi)$$
$$(\varphi \wedge \psi) \leftrightarrow \neg(\neg\varphi \vee \neg\psi)$$
- The above axiom schemata are jointly called the *standard axiom schemata* for Propositional Logic
 - They all are **valid**

Proofs

- A *proof* of a conclusion from a set of premises is a sequence of sentences terminating in the conclusion in which each item is either
 - (1) a premise,
 - (2) an instance of an axiom schema, or
 - (3) the result of applying a rule of inference to earlier items in sequence

- **Example:**

1. $p \rightarrow q$	Premise
2. $q \rightarrow r$	Premise
3. $(q \rightarrow r) \rightarrow (p \rightarrow (q \rightarrow r))$	II
4. $p \rightarrow (q \rightarrow r)$	MP : 3,2
5. $(p \rightarrow (q \rightarrow r)) \rightarrow ((p \rightarrow q) \rightarrow (p \rightarrow r))$	ID
6. $(p \rightarrow q) \rightarrow (p \rightarrow r)$	MP : 5,4
7. $p \rightarrow r$	MP : 6,1

Proofs (cont')

- If there exists a proof of a sentence φ from a set Δ of premises and the standard axiom schemata using Modus Ponens, then φ is said to be *provable* from Δ , written as

$$\Delta \vdash \varphi$$

- There is a close connection between provability and logical entailment (\models):

A set of sentences Δ logically entails a sentence φ
if and only if φ is provable from Δ

- Soundness Theorem:

If φ is provable from Δ , then Δ logically entails φ .

- Completeness Theorem:

If Δ logically entails φ , then φ is provable from Δ .

Proofs (cont')

- The concept of provability is important because it suggests how we can automate the determination of logical entailment
 - Starting from a set of premises Δ , we enumerate conclusions from this set
 - If a sentence φ appears, then it is provable from Δ and is, therefore, a logical consequence
 - If the negation of φ appears, then $\neg\varphi$ is a logical consequence of Δ and φ is not logically entailed (unless Δ is inconsistent)
 - Note that it is possible that neither φ nor $\neg\varphi$ will appear

Resolution

- **Propositional resolution** is an extremely powerful **rule of inference** for Propositional Logic
- Using propositional resolution (without axiom schemata or other rules of inference), it is possible to build a theorem prover that is **sound and complete** for all of Propositional Logic
- The search space using propositional resolution is much smaller than for standard propositional logic
- Propositional resolution works only on expressions in ***clausal form***
 - Before the rule can be applied, the premises and conclusions must be converted to this form

Clausal Forms

- A **clause** is a set of literals which is assumed (implicitly) to be a disjunction of those literals

– Example: $\neg p \vee q \vee \neg r \iff \{\neg p, q, \neg r\}$

- **Unit clause**: clause with only one literal; e.g. $\{\neg q\}$
- **Clausal form** of a formula: Implicit conjunction of clauses

- Example: $p \wedge (\neg p \vee q \vee \neg r) \wedge (\neg q \vee q \vee \neg r) \wedge (\neg q \vee p)$

\Updownarrow

$\{\{p\}, \{\neg p, q, \neg r\}, \{\neg q, q, \neg r\}, \{\neg q, p\}\}$

Abbreviated notation: $\{p, \bar{p}q\bar{r}, \bar{q}q\bar{r}, \bar{q}p\}$

- Notation:
 - l -literal, l^c -complement of l
 - C -clause (a set of literals)
 - S -a clausal form (a set of clauses)

Resolution – Properties of Clausal Forms

- (1) If l appears in some clause of S , but l^c does not appear in any clause, then, if we delete all clauses in S containing l , the new clausal form S' is satisfiable if and only if S is satisfiable

Example: Satisfiability of

is equivalent to $S = \{pq\bar{r}, p\bar{q}, \bar{p}q\}$ if

- (2) Suppose $C = \{l\}$ is a unit clause and we obtain $S' = \{p\bar{q}, \bar{p}q\}$ from S by deleting C and l^c from all clauses that contain it; then, S is satisfiable if and only if S' is satisfiable

Example:

$S = \{p, \bar{p}q\bar{r}, \bar{q}q\bar{r}, q\bar{p}\}$ is satisfiable iff
 $S' = \{q\bar{r}, \bar{q}q\bar{r}, q\}$ is satisfiable

Resolution – Properties of Clausal Forms (cont')

(3) If S contains two clauses C and C' , such that C is a subset of C' , we can delete C' without affecting the (un)satisfiability of S

Example: $S = \{p, \bar{p}q\bar{r}, \bar{q}q\bar{r}, q\bar{p}\}$ is satisfiable iff

$$S = \{p, \bar{p}q\bar{r}, \bar{q}q\bar{r}, q\bar{p}\} \text{ is satisfiable}$$

(4) If a clause C in $S = \{p, \bar{p}q\bar{r}, \bar{q}q\bar{r}, q\bar{p}\}$ contains a pair of complementary literals l, l^c , then C can be deleted from S without affecting its (un)satisfiability

Example: $S = \{p, \bar{p}q\bar{r}, \bar{q}q\bar{r}, q\bar{p}\}$ is satisfiable iff

$$S = \{p, \bar{p}q\bar{r}, \bar{q}q\bar{r}, q\bar{p}\} \text{ is such}$$

$$S' = \{p, \bar{p}q\bar{r}, q\bar{p}\}$$

Converting to clausal form

Theorem: Every propositional formula can be transformed into an equivalent formula in CNF

1. Implications:

$$\begin{aligned}\varphi_1 \rightarrow \varphi_2 &\rightarrow \neg\varphi_1 \vee \varphi_2 \\ \varphi_1 \leftarrow \varphi_2 &\rightarrow \varphi_1 \vee \neg\varphi_2 \\ \varphi_1 \leftrightarrow \varphi_2 &\rightarrow (\neg\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \neg\varphi_2)\end{aligned}$$

2. Negations:

$$\begin{aligned}\neg\neg\varphi &\rightarrow \varphi \\ \neg(\varphi_1 \wedge \varphi_2) &\rightarrow \neg\varphi_1 \vee \neg\varphi_2 \\ \neg(\varphi_1 \vee \varphi_2) &\rightarrow \neg\varphi_1 \wedge \neg\varphi_2\end{aligned}$$

3. Distribution:

$$\begin{aligned}\varphi_1 \vee (\varphi_2 \wedge \varphi_3) &\rightarrow (\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \varphi_3) \\ (\varphi_1 \wedge \varphi_2) \vee \varphi_3 &\rightarrow (\varphi_1 \vee \varphi_3) \wedge (\varphi_2 \vee \varphi_3) \\ (\varphi_1 \vee \varphi_2) \vee \varphi_3 &\rightarrow \varphi_1 \vee (\varphi_2 \vee \varphi_3) \\ (\varphi_1 \wedge \varphi_2) \wedge \varphi_3 &\rightarrow \varphi_1 \wedge (\varphi_2 \wedge \varphi_3)\end{aligned}$$

4. Operators:

$$\begin{aligned}\varphi_1 \vee \dots \vee \varphi_n &\rightarrow \{\varphi_1, \dots, \varphi_n\} \\ \varphi_1 \wedge \dots \wedge \varphi_n &\rightarrow \{\varphi_1\} \dots \{\varphi_n\}\end{aligned}$$

Example

- Transform the formula

$$(p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p)$$

into an equivalent formula in CNF

Solution:

$$(p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p)$$

$$\equiv (\neg p \vee q) \rightarrow (\neg\neg q \vee \neg p)$$

$$\equiv \neg(\neg p \vee q) \vee (\neg\neg q \vee \neg p)$$

$$\equiv (\neg\neg p \wedge \neg q) \vee (\neg\neg q \vee \neg p)$$

$$\equiv (p \wedge \neg q) \vee (q \vee \neg p)$$


$$\equiv (p \vee q \vee \neg p) \wedge (\neg q \vee q \vee \neg p)$$

Resolution Rule

- Suppose C_1, C_2 are clauses such that l in C_1, l^c in C_2 . The clauses C_1 and C_2 are said to be **clashing clauses** and they clash on the complementary literals l, l^c

C , the **resolvent** of C_1, C_2 is the clause

$$Res(C_1, C_2) = (C_1 - \{l\}) \cup (C_2 - \{l^c\})$$

C_1 and C_2 are  **t clauses of C.**

$$C = (C_1 - \{l\}) \cup (C_2 - \{l^c\})$$

Resolution Rule (cont')

- Example:

The clauses

$$C_1 = \bar{p}q\bar{r}, \quad C_2 = \bar{q}p$$

clash on p, \bar{p} .

$$Res(C_1, C_2) = q\bar{r} \cup \bar{q} = q\bar{r}\bar{q}$$

C_1, C_2 also clash on q, \bar{q} , so, another way to find

a resolvent for $Res(C_1, C_2) = \bar{p}\bar{r} \cup p = \bar{p}\bar{r}p$ is

Resolution (cont')

- Theorem: Resolvent C is satisfiable if and only if the parent clauses C_1, C_2 are simultaneously satisfiable

- Resolution Algorithm:

Input: S – a set of clauses

Output: “ S is satisfiable” or “ S is not satisfiable”

1. Set $S_0 := S$
2. Suppose S_i has already been constructed
3. To construct S_{i+1} , choose a pair of clashing literals and clauses C_1, C_2 in S (if there are any) and derive

$$C := \text{Res}(C_1, C_2)$$

$$S_{i+1} := S_i \cup \{C\}$$

1. If C is the empty clause, output “ S is not satisfiable”; if $S_{i+1} = S_i$, output “ S is satisfiable”
2. Otherwise, set $i := i + 1$ and go back to Step 2

Resolution (cont')

- Example: Show that $(p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p)$ is a valid formula

Solution: We will show that

$$\neg[(p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p)]$$

is not satisfiable.

(1) Transform the formula into CNF:

$$\begin{aligned}\neg[(p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p)] &\equiv (p \rightarrow q) \wedge \neg(\neg q \rightarrow \neg p) \\ &\equiv (\neg p \vee q) \wedge \neg q \wedge \neg\neg p \\ &\equiv (\neg p \vee q) \wedge \neg q \wedge p\end{aligned}$$

(2) Show, using resolution, that

$$S = \{\bar{p}q, \bar{q}, p\}$$

1.

2. $S_0 = \{\bar{p}q, \bar{q}, p\}$

3. $C_1 = \bar{p}q, C_2 = \bar{q}, C = \bar{p}, S_1 = \{\bar{p}q, \bar{q}, p, \bar{p}\} C_1 = p, C_2 = \bar{p},$

- A derivation of the empty clause from S is called a **refutation** of S

Resolution (cont')

- Theorem: *If the set of a clauses labeling the leaves of a resolution tree is satisfiable, then the clause at the root is satisfiable*
- Theorem (*Soundness*): *If the empty clause is derived from a set of clauses, then the set of clauses is unsatisfiable*
- Theorem (*Completeness*) *If a set of clauses is unsatisfiable, then the empty clause can be derived from it using resolution algorithm*

ILLUSTRATION BY LARGER
EXAMPLE

Problem Example

- For each of these sets of premises, what relevant conclusion or conclusions can be drawn? Explain the rules of inference used to obtain each conclusion from the premises.
 - (a) “If I eat spicy foods, then I have strange dreams.” “I have strange dreams if there is thunder while I sleep.” “I did not have strange dreams.”
 - (b) “I am dreaming or hallucinating.” “I am not dreaming.” “If I am hallucinating, I see elephants running down the road.”
 - (c) “If I work, it is either sunny or partly sunny.” “I worked last Monday or I worked last Friday.” “It was not sunny on Tuesday.” “It was not partly sunny on Friday.”

Solution (a)

(a) “If I eat spicy foods, then I have strange dreams.” “I have strange dreams if there is thunder while I sleep.” “I did not have strange dreams.”

- The relevant conclusions are: “I did not eat spicy food” and “There is no thunder while I sleep”.

- Let the primitive statements be:

- s , ‘I eat spicy foods’
- d , ‘I have strange dreams’
- t , ‘There is thunder while I sleep’

- Then the premises are translated as: $s \rightarrow d$, $t \rightarrow d$, and $\neg d$.

- And the conclusions: $\neg s$, $\neg t$.

- Steps Reason

1. $s \rightarrow d$ *premise*

2. $\neg d$ *premise*

3. $\neg s$ *Modus Tollens to Steps 1 and 2*

4. $t \rightarrow d$ *premise*

5. $\neg t$ *Modus Tollens to Steps 4 and 2.*

Solution (b)

(b) “I am dreaming or hallucinating.” “I am not dreaming.” “If I am hallucinating, I see elephants running down the road.”

- The relevant conclusion is: “I see elephants running down the road.”
- Let the primitive statements be:
 - d , ‘I am dreaming’
 - h , ‘I am hallucinating’
 - e , ‘I see elephants running down the road’
- Then the premises are translated as: $d \vee h$, $\neg d$, and $h \rightarrow e$.
- And the conclusion: e .
- Steps Reason
 1. $d \vee h$ *premise*
 2. $\neg d$ *premise*
 3. h *rule of disjunctive syllogism to Steps 1 and 2*
 4. $h \rightarrow e$ *premise*
 5. e *Modus Ponens to Steps 4 and 3*

Solution (c)

- (c) “If I work, it is either sunny or partly sunny.” “I worked last Monday or I worked last Friday.” “It was not sunny on Tuesday.” “It was not partly sunny on Friday.”
- There is no single relevant conclusion in this problem, its main difficulty is to to represent the premises so that one is able infer anything at all. One possible relevant conclusion is: “It was sunny or partly sunny last Monday or it was sunny last Friday.”
 - Let the primitive statements be:
 - wm , ‘I worked last Monday’
 - wf , ‘I worked last Friday’
 - sm , ‘It was sunny last Monday’
 - st , ‘It was sunny last Tuesday’
 - sf , ‘It was sunny last Friday’
 - pm , ‘It was partly sunny last Monday’
 - pf , ‘It was partly sunny last Friday’
 - Then the premises are translated as: $wm \vee wf$, $wm \rightarrow (sm \vee pm)$, $wf \rightarrow (sf \vee pf)$, $\neg st$, and $\neg pf$.
 - And the conclusion: $sf \vee sm \vee pm$.

Solution (c) – Method 1

- Steps Reason
- 1. $wf \rightarrow (sf \vee pf)$ premise
- 2. $\neg wf \vee sf \vee pf$ expression for implication
- 3. $\neg pf \rightarrow (\neg wf \vee sf)$ expression for implication
- 4. $\neg pf$ premise
- 5. $\neg wf \vee sf$ modus ponens to Steps 3 and 4
- 6. $wf \rightarrow sf$ expression for implication
- 7. $wm \vee wf$ premise
- 8. $\neg wm \rightarrow wf$ expression for implication
- 9. $\neg wm \rightarrow sf$ rule of syllogism to Steps 8 and 6
- 10. $wm \vee sf$ expression for implication
- 11. $\neg sf \rightarrow wm$ expression for implication
- 12. $wm \rightarrow (sm \vee pm)$ premise
- 13. $\neg sf \rightarrow (sm \vee pm)$ rule of syllogism to Steps 11 and 12
- 14. $sf \vee sm \vee pm$ expression for implication.

Solution (c) – Method 2 (Use the rule of resolution)

- Steps Reason
- 1. $wf \rightarrow (sf \vee pf)$ premise
- 2. $\neg wf \vee sf \vee pf$ expression for implication
- 3. $\neg pf$ premise
- 4. $\neg wf \vee sf$ rule of resolution to Steps 2 and 3
- 5. $wm \vee wf$ premise
- 6. $wm \vee sf$ rule of resolution to Steps 4 and 5
- 7. $wm \rightarrow (sm \vee pm)$ premise
- 8. $\neg wm \vee sm \vee pm$ expression for implication
- 9. $sf \vee sm \vee pm$ rule of resolution to Steps 7 and 8

Rule-Based Deduction Systems

The way in which a piece of knowledge is expressed by a human expert carries important information,

example: if the person has fever and feels tummy-pain then she may have an infection.

In logic it can be expressed as follows:

$$\forall x. (\text{has_fever}(x) \ \& \ \text{tummy_pain}(x) \ \rightarrow \ \text{has_an_infection}(x))$$

If we convert this formula to **clausal form** we lose the content as then we may have equivalent formulas like:

$$\begin{aligned} \text{(i)} \quad & \text{has_fever}(x) \ \& \ \sim\text{has_an_infection}(x) \ \rightarrow \ \sim\text{tummy_pain}(x) \\ \text{(ii)} \quad & \sim\text{has_an_infection}(x) \ \& \ \text{tummy_pain}(x) \ \rightarrow \\ & \sim\text{has_fever}(x) \end{aligned}$$

Notice that:

- (i) and (ii) are **logically equivalent** to the original sentence
- they have **lost the main information** contained in its formulation.

Forward production systems

- The main idea behind the forward/backward production systems is:
 - to take advantage of the implicational form in which production rules are stated by the expert
 - and use that information to help achieving the goal.
- In the present systems the formulas have two forms:
 - rules
 - and facts

Forward production systems

- Rules are the **productions** stated in implication form.
 - Rules express **specific knowledge** about the problem.
 - Facts are assertions not expressed as implications.
 - The task of the system will be to prove a **goal formula** with these facts and rules.
 - In a **forward production system** the rules are expressed as *F-rules*
 - **F-rules** operate on the **global database** of facts until the termination condition is achieved.
 - This sort of proving system is a *direct system* rather than a *refutation system*.
- Facts
 - Facts are expressed in **AND/OR form**.
 - An expression in AND/OR form consists on sub-expressions of literals connected by & and V symbols.
 - An expression in AND/OR form **is not in clausal form**.

Forward production systems

Rule-Based Deduction Systems

Steps to transform facts into AND/OR form for forward system:

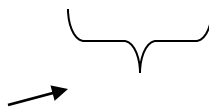
1. *Eliminate* (temporarily) implication symbols.
2. Reverse quantification of variables in first disjunct by *moving negation symbol*.
3. *Skolemize* existential variables.
4. Move all *universal quantifiers* to the front and drop.
5. *Rename variables* so the same variable does not occur in different main conjuncts
 - Main conjuncts are small AND/OR trees, not necessarily sum of literal clauses as in Prolog.

EXAMPLE

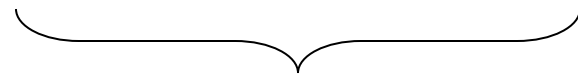
Original formula: $\exists u. \forall v. \{q(v, u) \& \sim[[r(v) \vee p(v)] \& s(u, v)]\}$

converted formula: $q(w, a) \& \{[\sim r(v) \& \sim p(v)] \vee \sim s(a, v)\}$

Conjunction of two
main conjuncts



Various variables in conjuncts



All variables appearing on the final expressions are assumed to be **universally quantified**.

Rule-Based Deduction Systems: forward production systems

F-rules

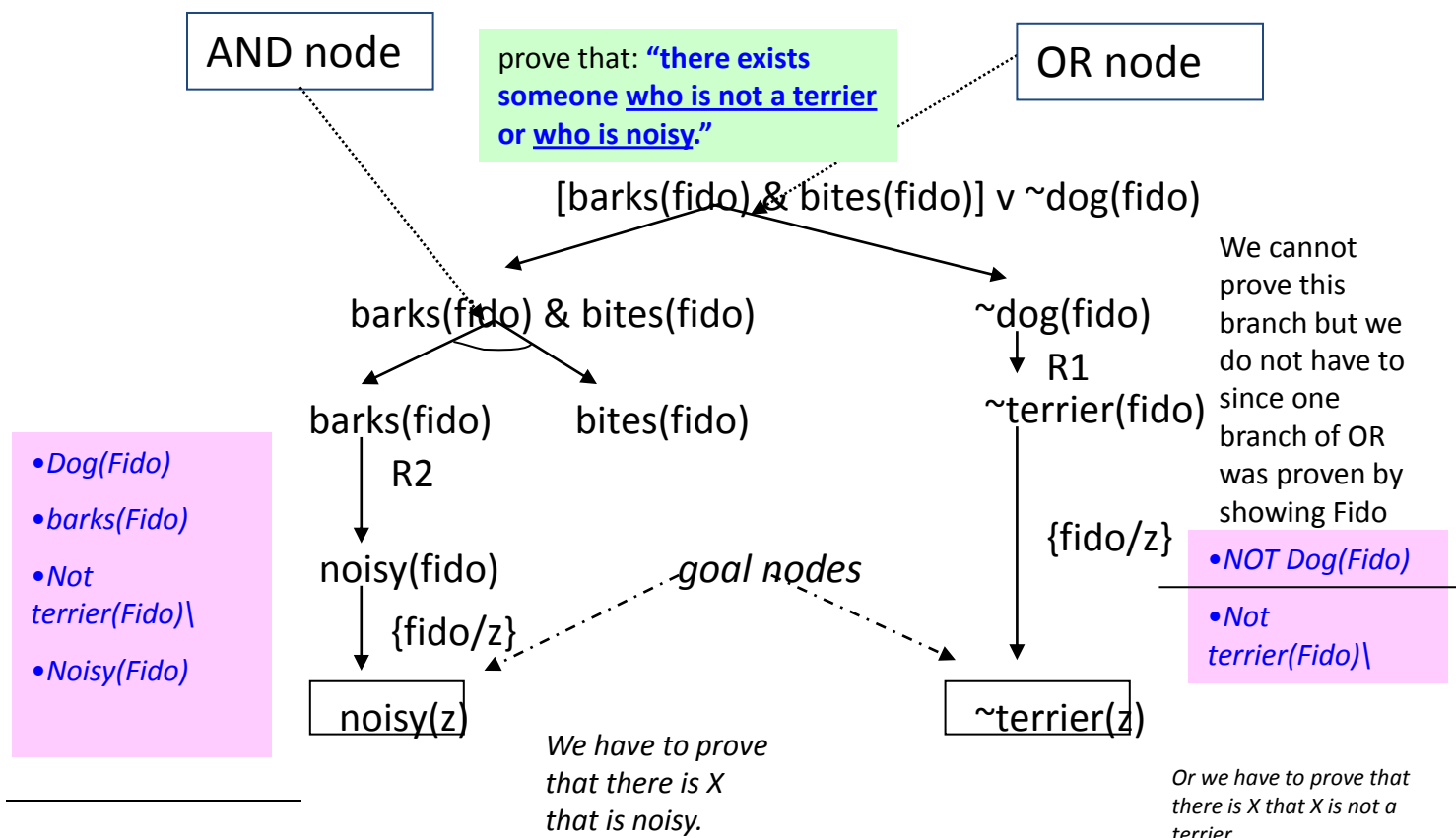
Rules in a **forward production system** will be applied to the AND/OR graph to produce new transformed graph structures.

We assume that rules in a forward production system are of the form:

$$L \implies W,$$

where L is a literal and W is a formula in AND/OR form.

- Recall that a rule of the form $(L1 \vee L2) \implies W$ is equivalent to the pair of rules: $L1 \implies W \vee L2 \implies W$.



forward production systems

Steps to transform the rules into a [free-quantifier form](#):

1. Eliminate (temporarily) implication symbols.
2. Reverse quantification of variables in first disjunct by moving negation symbol.
3. Skolemize existential variables.
4. Move all universal quantifiers to the front and drop.
5. Restore implication.

All variables appearing on the final expressions are assumed to be universally quantified.

E.g. Original formula: $\forall x. (\exists y. \forall z. (p(x, y, z)) \rightarrow \forall u. q(x, u))$

Converted formula: $p(x, y, f(x, y)) \rightarrow q(x, u).$

Skolem
function

Restored
implication

forward production systems

A full *example*:

- Fact: Fido barks and bites, or Fido is not a dog.
- (R1) All terriers are dogs.
- (R2) Anyone who barks is noisy.

Based on these facts, prove that: “**there exists someone who is not a terrier or who is noisy.”**”

Logic representation:

↖
goal

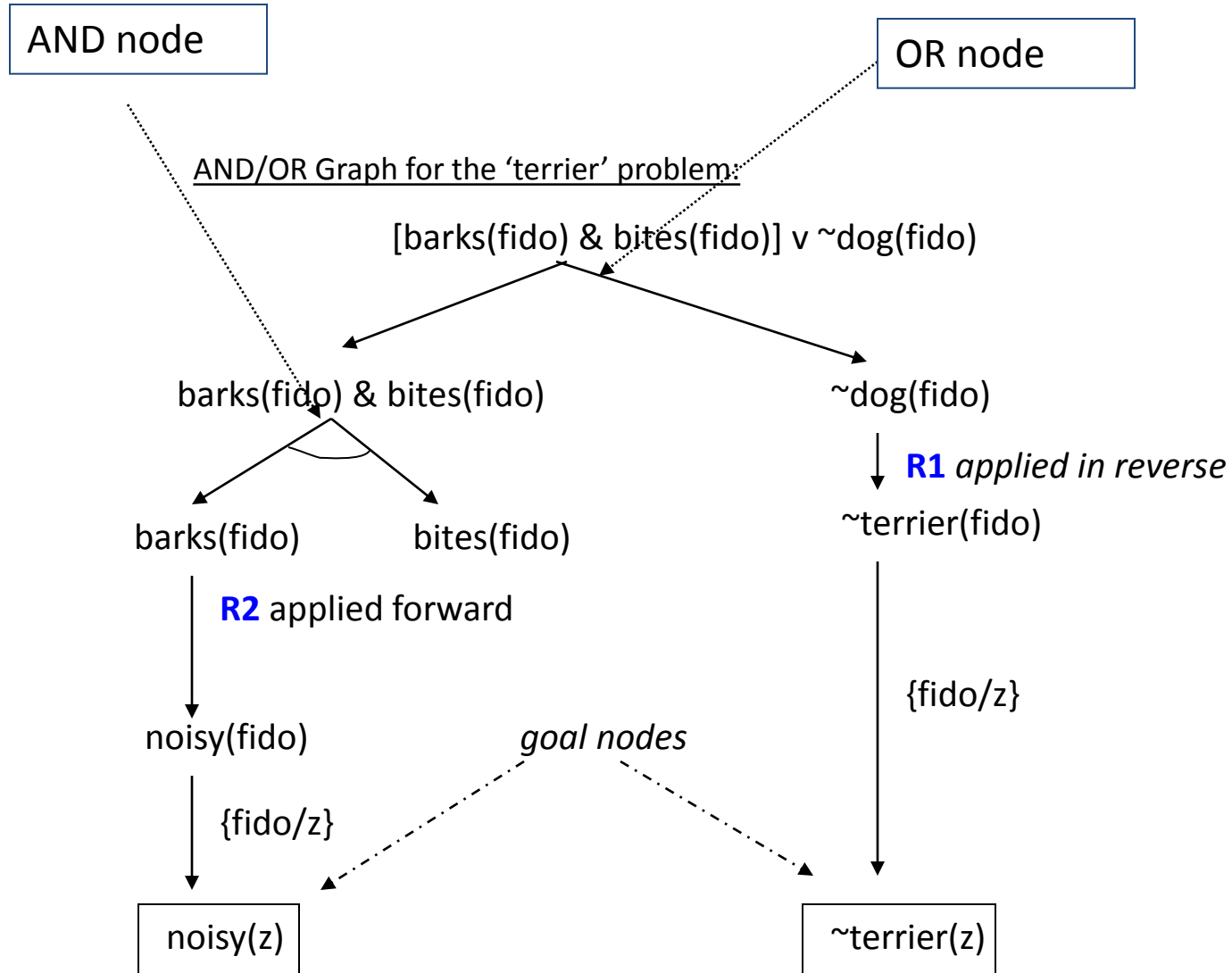
$(\text{barks}(\text{fido}) \ \& \ \text{bites}(\text{fido})) \ \vee \ \sim\text{dog}(\text{fido})$

R1: $\text{terrier}(x) \rightarrow \text{dog}(x)$

R2: $\text{barks}(y) \rightarrow \text{noisy}(y)$

goal: $\exists w. (\sim\text{terrier}(w) \vee \text{noisy}(w))$

From facts to goal



Backward production systems

B-Rules

We restrict B-rules to expressions of the form: $W \implies L$,
where W is an expression in AND/OR form and L is a literal,
and the **scope of quantification** of any variables in the implication **is the entire implication**.

Recall that $W \implies (L1 \ \& \ L2)$ is equivalent to the two rules: $W \implies L1$ and $W \implies L2$.

An important property of logic is the **duality** between **assertions** and **goals** in theorem-proving systems.

Duality between assertions and goals allows the **goal expression** to be treated as **if it were an assertion**.

Conversion of the goal expression into AND/OR form:

1. Elimination of implication symbols.
2. Move negation symbols in.
3. Skolemize *existential* variables.
4. Drop existential quantifiers. Variables remaining in the AND/OR form are considered to be existentially quantified.

Goal clauses are **conjunctions of literals** and the **disjunction of these clauses** is the **clause form** of the goal well-formed formula.

Example 1 of formulation of Rule-Based Deduction Systems

1. Facts:

- dog(fido)
- ~barks(fido)
- wags-tail(fido)
- meows(myrtle)

Rules:

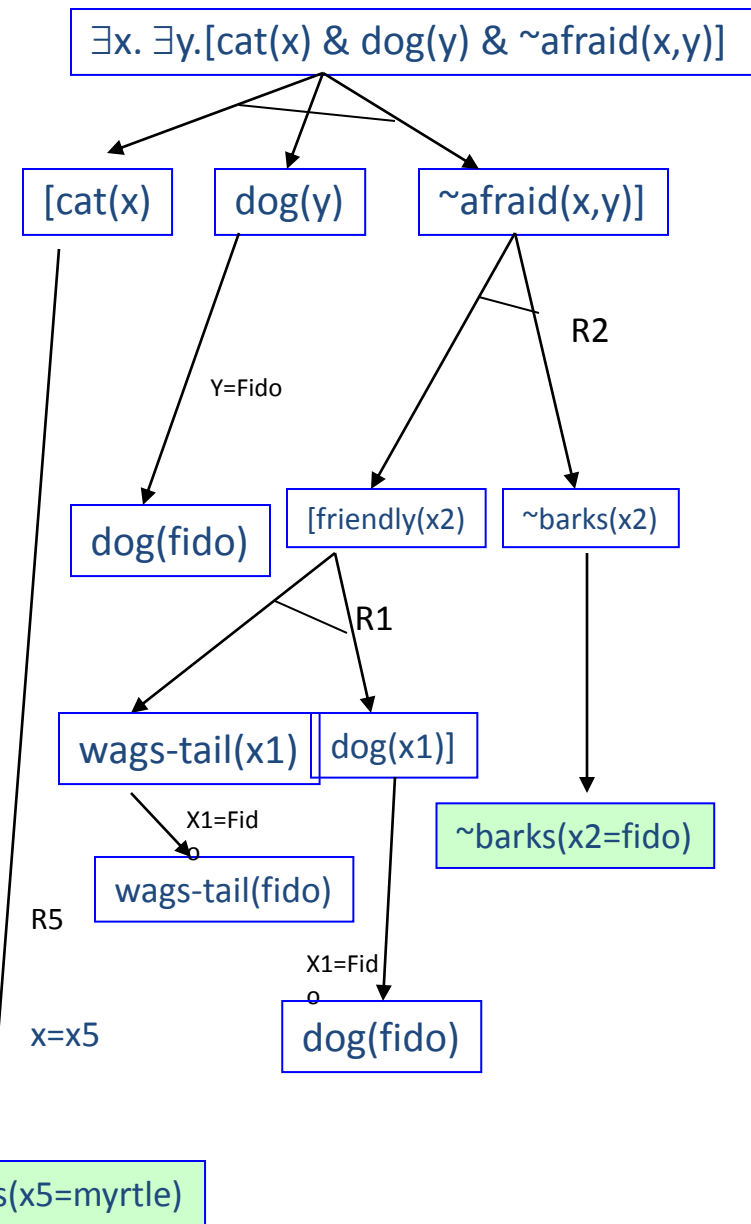
- R1: [wags-tail(x1) & dog(x1)] → friendly(x1)
- R2: [friendly(x2) & ~barks(x2)] → ~afraid(y2,x2)
- R3: dog(x3) → animal(x3)
- R4: cat(x4) → animal(x4)
- R5: meows(x5) → cat(x5)

Suppose we want to ask if there are a cat and a dog such that the cat is unafraid of the dog.

The goal expression is:

$$\exists x. \exists y. [\text{cat}(x) \ \& \ \text{dog}(y) \ \& \ \sim\text{afraid}(x,y)]$$

We treat the goal expression as an assertion



Rule-Based Deduction Systems

2. The blocks-world situation is described by the following set of wffs:

on_table(a)	clear(e)
on_table(c)	clear(d)
on(d,c)	heavy(d)
on(b,a)	wooden(b)
heavy(b)	on(e,b)

Homework:

formulation of Rule-Based
Deduction Systems

The following statements provide general knowledge about this blocks world:

Every big, blue block is on a green block.

Each heavy, wooden block is big.

All blocks with clear tops are blue.

All wooden blocks are blue.

Represent these statements by a set of implications having single-literal consequents.

Draw a consistent AND/OR solution tree (using B-rules) that solves the problem: “Which block is on a green block?”

HOMEWORK Problem 2.

Transformation of rules and goal:

Facts:

f1: on_table(a)	f6: clear(e)
f2: on_table(c)	f7: clear(d)
f3: on(d,c)	f8: heavy(d)
f4: on(b,a)	f9: wooden(b)
f5: heavy(b)	f10: on(e,b)

Rules:

R1: $\text{big}(y1) \wedge \text{blue}(y1) \rightarrow \text{green}(g(y1))$ Every big, blue block is on a green block.

R2: $\text{big}(y0) \wedge \text{blue}(y0) \rightarrow \text{on}(y0,g(y0))$ “ “ “ “ “ “ “ “

R3: $\text{heavy}(z) \wedge \text{wooden}(z) \rightarrow \text{big}(z)$ Each heavy, wooden block is big.

R4: $\text{clear}(x) \rightarrow \text{blue}(x)$ All blocks with clear tops are blue.

R5: $\text{wooden}(w) \rightarrow \text{blue}(w)$ All wooden blocks are blue.

Goal:

$\text{green}(u) \wedge \text{on}(v,u)$ Which block is on a green block?

HOMEWORK PROBLEM 3. Information

Retrieval System

- We have a set of facts containing personnel data for a business organization
- and we want an automatic system to answer various questions about personal matters.

- Facts

John Jones is the manager of the Purchasing Department.

```
manager(p-d,john-jones)
works_in(p-d, joe-smith)
works_in(p-d,sally-jones)
works_in(p-d,pete-swanson)
```

Harry Turner is the manager of the Sales Department.

```
manager(s-d,harry-turner)
works_in(s-d,mary-jones)
works_in(s-d,bill-white)
married(john-jones,mary-jones)
```

Rule-Based Deduction Systems

Rules

- R1: $\text{manager}(x,y) \rightarrow \text{works_in}(x,y)$
- R2: $\text{works_in}(x,y) \ \& \ \text{manager}(x,z) \rightarrow \text{boss_of}(y,z)$
- R3: $\text{works_in}(x,y) \ \& \ \text{works_in}(x,z) \rightarrow \sim \text{married}(y,z)$
- R4: $\text{married}(y,z) \rightarrow \text{married}(z,y)$
- R5: $[\text{married}(x,y) \ \& \ \text{works_in}(p-d,x)] \rightarrow \text{insured_by}(x,\text{eagle-corp})$

In this company
married people should
not work in the same
department

With these facts and rules a **simple backward production system** can answer a variety of questions.

Build solution graphs for the following questions:

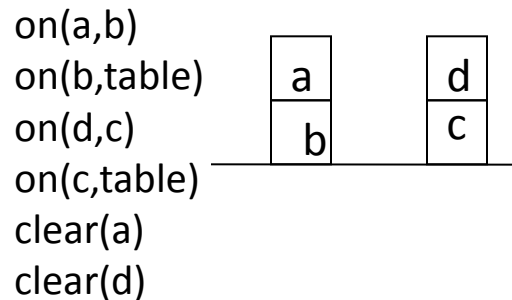
1. Name someone who works in the Purchasing Department.
2. Name someone who is married and works in the sales department.
3. Who is Joe Smith's boss?
4. Name someone insured by Eagle Corporation.
5. Is John Jones married with Sally Jones?

Planning

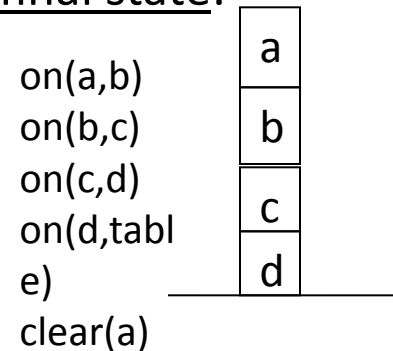
- **Planning** is fundamental to “intelligent” behavior. E.g.
 - assembling tasks
 - route finding
 - planning chemical processes
 - planning a report
- **Representation**

The planner has to represent states of the world it is operating within, and to predict consequences of carrying actions in its world. E.g.

initial state:



final state:



Planning

- Representing an action

One standard method is by specifying sets of preconditions and effects, e.g.

pickup(X) :

preconditions: clear(X), handempty.

deletelist: on(X,_), clear(X), handempty.

addlist: holding(X).

Planning

- The Frame Problem in Planning
- This is the problem of how to keep track in a representation of the world of all the effects that an action may have.
- The action representation given is the one introduced by STRIPS (Nilsson) and is an **attempt to** a solution to the **frame problem**
 - but it is only adequate for simple actions in simple worlds.

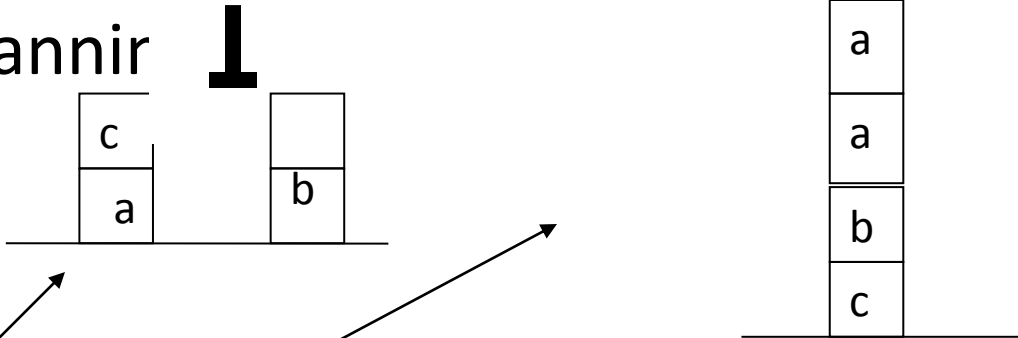
- The Frame Axiom
- The frame axiom states that a fact is true if it is **not in the last delete list** and was **true in the previous state**.
- The frame axiom states that a fact is false if it is **not in the last add list** and was **false in the previous state**.

Planning

- Control Strategies
 - Forward Chaining
 - Backward Chaining

The choice on which of these strategies to use depends on the problem, normally backward chaining is more effective.

Plannir



Example:

Initial State

clear(b), clear(c), on(c,a), ontable(a), ontable(b), handempty

Goal

on(b,c) & on(a,b)

Rules

R1: pickup(x)

P & D: ontable(x), clear(x),

handempty

A: holding(x)

R2: putdown(x)

P & D: holding(x)

A: ontable(x), clear(x), handempty

R3: stack(x,y)

P & D: holding(x), clear(y)

A: handempty, on(x,y), clear(x)

R4: unstack(x,y)

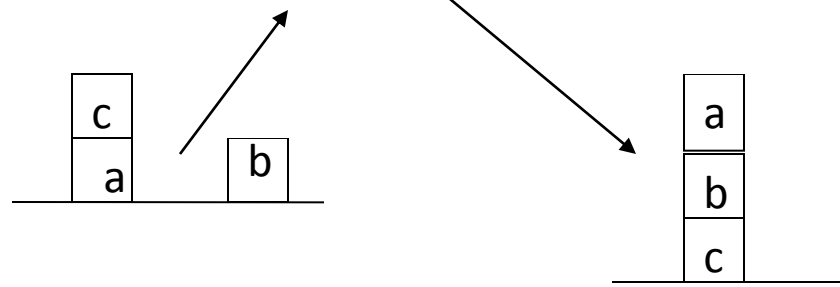
P & D: on(x,y), clear(x), handempty

A: holding(x), clear(y)

handempty	A: ontable(x), clear(x), handempty
A: holding(x)	
R3: stack(x,y)	R4: unstack(x,y)
P & D: holding(x), clear(y)	P & D: on(x,y), clear(x), handempty
A: handempty, on(x,y), clear(x)	A: holding(x), clear(y)

Planning

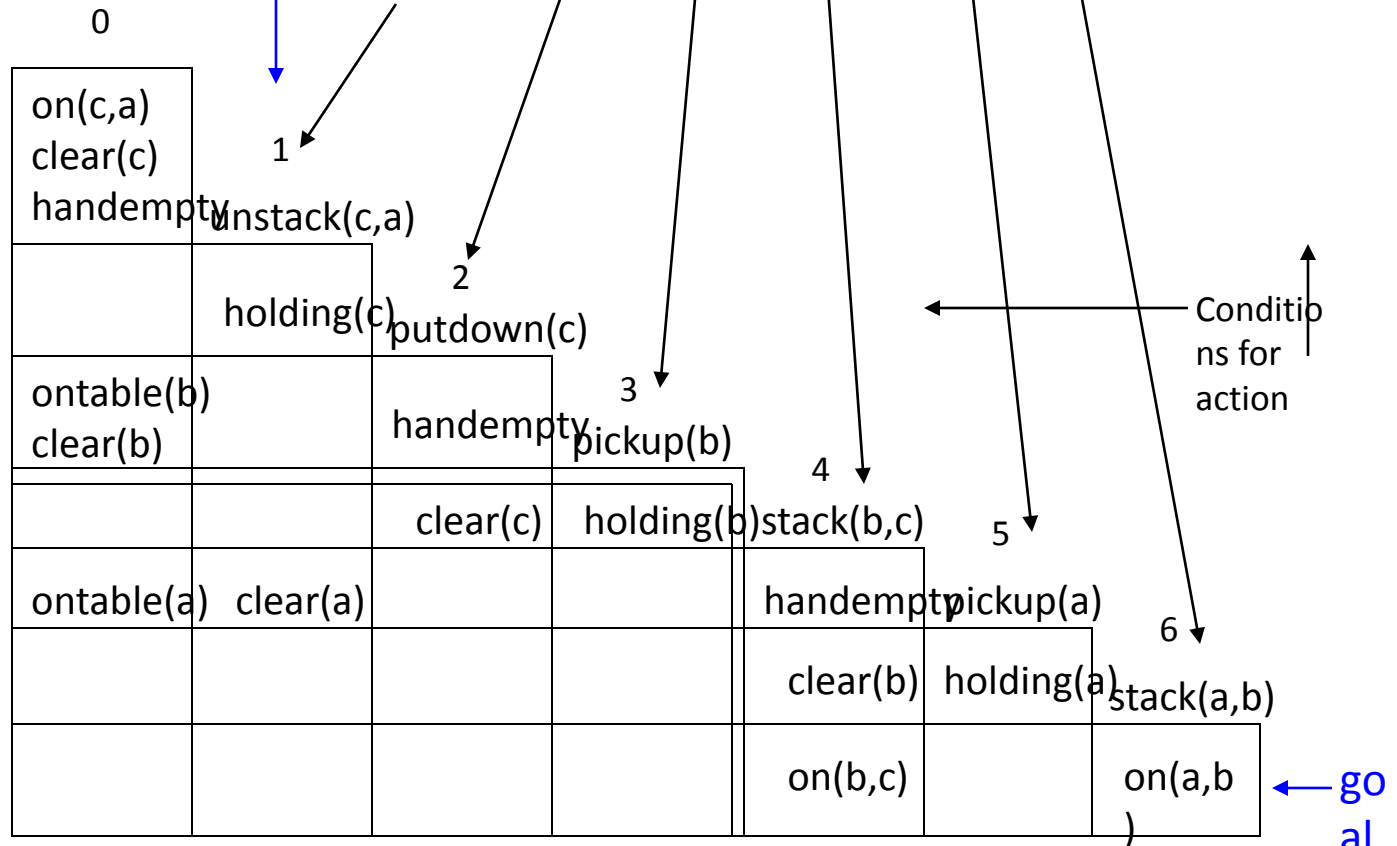
{unstack(c,a), putdown(c), pickup(b), stack(b,c), pickup(a), stack(a,b)}



Initial situation

next situation

TRIANGLE TABLE {unstack(c,a), putdown(c), pickup(b), stack(b,c), pickup(a), stack(a,b)}



Assignment Questions (Unit -2)

Ques 1: Describe how the two SCRIPS rules pickup(x) and stack(x,y) could be combined into a macro-rule put(x,y).

What are the preconditions, delete list and add list of the new rule.

Can you specify a general procedure for creating macro-rules components?

Ques 2: Consider the problem of devising a plan for a kitchen-cleaning robot.

(i) Write a set of STRIPS-style operators that might be used.

When you describe the operators, take into account the following considerations:

- (a) Cleaning the stove or the refrigerator will get the floor dirty.
- (b) The stove must be clean before covering the drip pans with tin foil.
- (c) Cleaning the refrigerator generates garbage and messes up the counters.
- (d) Washing the counters or the floor gets the sink dirty.

(ii) Write a description of an initial state of a kitchen that has a dirty stove, refrigerator, counters, and floor.

(The sink is clean, and the garbage has been taken out).

Also write a description of the goal state where everything is clean, there is no trash, and the stove drip pans have been covered with tin foil.

Assignment Questions (Unit -2)

Ques 3: Explain the following types of Knowledge:

- i. Domain Specific Knowledge
- ii. Common Sense Knowledge

Ques 4. Explain the inferencing rules in propositional logic. Solve the following problem with the help of these rules:

Test the validity of following argument:

“If milk is black, then every cow is white. If every cow is white then, it has four legs. If every cow has four legs, then every buffalo is white and brisk. The milk is black. Therefore, the buffalo is white”.

Ques 5: Explain Rule based deduction system, in detail.

Ques 6. Define probability. Differentiate between conditional and Unconditional Probability