Artificial Intelligence & Expert System

UNIT II

Knowledge Representation (KR)

Introduction to KR

Representation of knowledge and the reasoning process are central to the entire field of artificial intelligence.

The primary component of a knowledge-based agent is its knowledge-base.

A knowledge-base is a set of sentences.

Each sentence is expressed in a language called the knowledge representation language.

Introduction to KR

Sentences represent some assertions about the world.

There must mechanisms to derive new sentences from old ones.

This process is known as inference or reasoning.

Inference must obey the primary requirement that the new sentences should follow logically from the previous ones.

Introduction to KR

Logic is the primary vehicle for representing and reasoning about knowledge.

A logic consists of two parts, a language and a method of reasoning. The logical language, in turn, has two aspects, syntax and semantics:

Syntax specifies the symbols in the language and how they can be combined to form sentences. Hence facts about the world are represented as sentences in logic.

Introduction to KR

Semantics: It specifies what facts in the world a sentence refers to. Hence, also specifies how you assign a truth value to a sentence based on its meaning in the world.

A **fact** is a claim about the world, and may be true or false.

Syntactic Inference Method: The rules for determining a subset of logical expressions, called theorems of the logic. It refers to mechanical method for computing (deriving) new (true) sentences from existing sentences.

Introduction to KR

Facts are claims about the world that are True or False, whereas a **representation** is an expression (sentence) in some language that can be encoded in a computer program and stands for the objects and relations in the world. We need to ensure that the representation is consistent with reality, so that the following figure holds:

$$\begin{tikzcd}
\text{Representation:} & \text{Sentences} \arrow[r, "\text{entails}"] \arrow[d, "\text{Semantics refer to}"] & \text{Sentences} \arrow[d, "\text{Semantics refer to}"] \\
\text{World:} & \text{Facts} \arrow[r, "\text{follows}"] & \text{Facts}
\end{tikzcd}$$

Knowledge agent

An agent perceives its environment through sensors.

The complete set of inputs at a given time is called a percept.

The current percept, or a sequence of percepts can influence the actions of an agent.

The agent can change the environment through actuators or effectors.

An operation involving an effector is called an action. Actions can be grouped into action sequences.

Knowledge agent

Examples of Agents:

Humans can be looked upon as agents. They have eyes, ears, skin, taste buds, etc. for sensors; and hands, fingers, legs, mouth for effectors.

Robots are agents. Robots may have camera, sonar, infrared, bumper, etc. for sensors. They can have grippers, wheels, lights, speakers, etc. for actuators.

We also have software agents or softbots that have some functions as sensors and some functions as actuators. Askjeeves.com is an example of a softbot.

Knowledge agent

Intelligent agents should have capacity for:

Perceiving, that is, acquiring information from environment.

Knowledge Representation, that is, representing its understanding of the world.

Reasoning, that is, inferring the implications of what it knows and of the choices it has.

Acting, that is, choosing what it want to do and carry it out.

Predicate logic

Predicate logic is also known as first order logic.

symbols, or alphabets are the basic constructs:

Logical Symbols are symbols that have a standard meaning, like: AND, OR, NOT, ALL, EXISTS, IMPLIES, IFF, FALSE, =

Non-Logical Symbols are:

- Constants
  - Predicates: 1-ary, 2-ary, .., n-ary.
  - Functions: 0-ary, 1-ary, 2-ary, .., n-ary.
- Variables

Predicate logic

0-ary functions are also called individual constants.

predicates return true or false, where as functions can return any value.

One needs to be able to distinguish the identifiers used for predicates, functions, and variables by using some appropriate convention, for example, capitals for function and predicate symbols and lower cases for variables.

Terms: A Term is either an individual constant (a 0-ary function), or a variable, or an n-ary function applied to n terms: F(t1 t2 ..tn)

Predicate logic

An **Atomic Formula** is either FALSE or an n-ary predicate applied to n terms: P(t1 t2 .. tn). In the case that "=" is a logical symbol in the language, (t1 = t2), where t1 and t2 are terms, is an atomic formula.

A **Literal** is either an atomic formula (a Positive Literal), or the negation of an atomic formula (a Negative Literal). A Ground Literal is a variable-free literal.

A **Clause** is a disjunction of literals. A Ground Clause is a variable-free clause. A Horn Clause is a clause with at most one positive literal. A Definite Clause is a Horn Clause with exactly one positive Literal.

WFF: Well Formed Formula

A Formula is either:

- an atomic formula, or
- a Negation, i.e. the NOT of a formula, or
- a Conjunctive Formula, i.e. the AND of formulae, or
- a Disjunctive Formula, i.e. the OR of formulae, or
- an Implication, that is a formula of the form (formula1 IMPLIES formula2), or

WFF: Well Formed Formula

- an Equivalence, that is a formula of the form (formula1 IFF formula2), or

- a Universally Quantified Formula, that is a formula of the form (ALL variable formula). We say that occurrences of variable are bound in formula [we should be more precise]. Or

- a Existentially Quantified Formula, that is a formula of the form (EXISTS variable formula). We say that occurrences of variable are bound in formula [we should be more precise].

WFF: Well Formed Formula

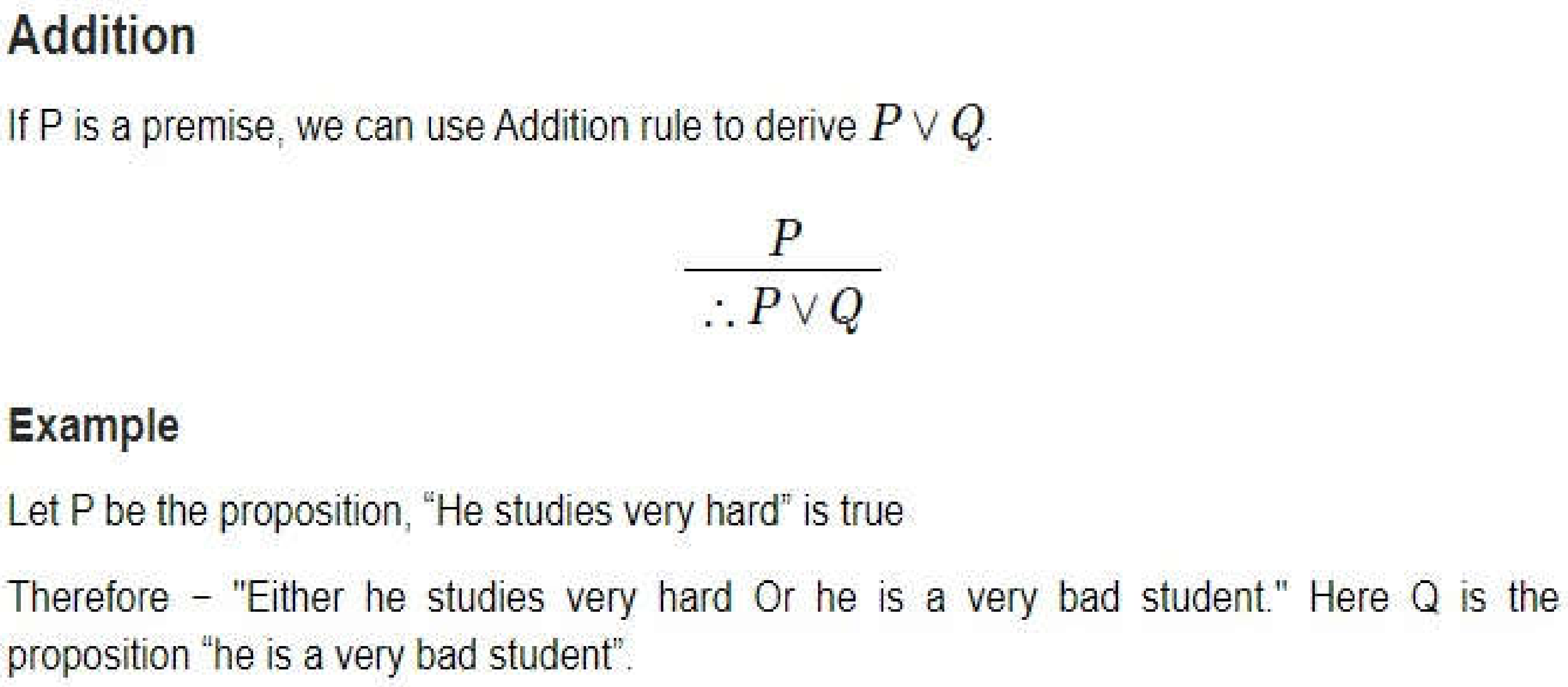
An occurrence of a variable in a formula that is not bound, is said to be **free.**

A formula where all occurrences of variables are bound is called a **closed formula.**

one where all variables are free is called an **open formula**.

A formula that is the disjunction of clauses is said to be in **Clausal Form**.

Inference rule & theorem proving



Addition

If P is a premise, we can use Addition rule to derive $P \lor Q$.

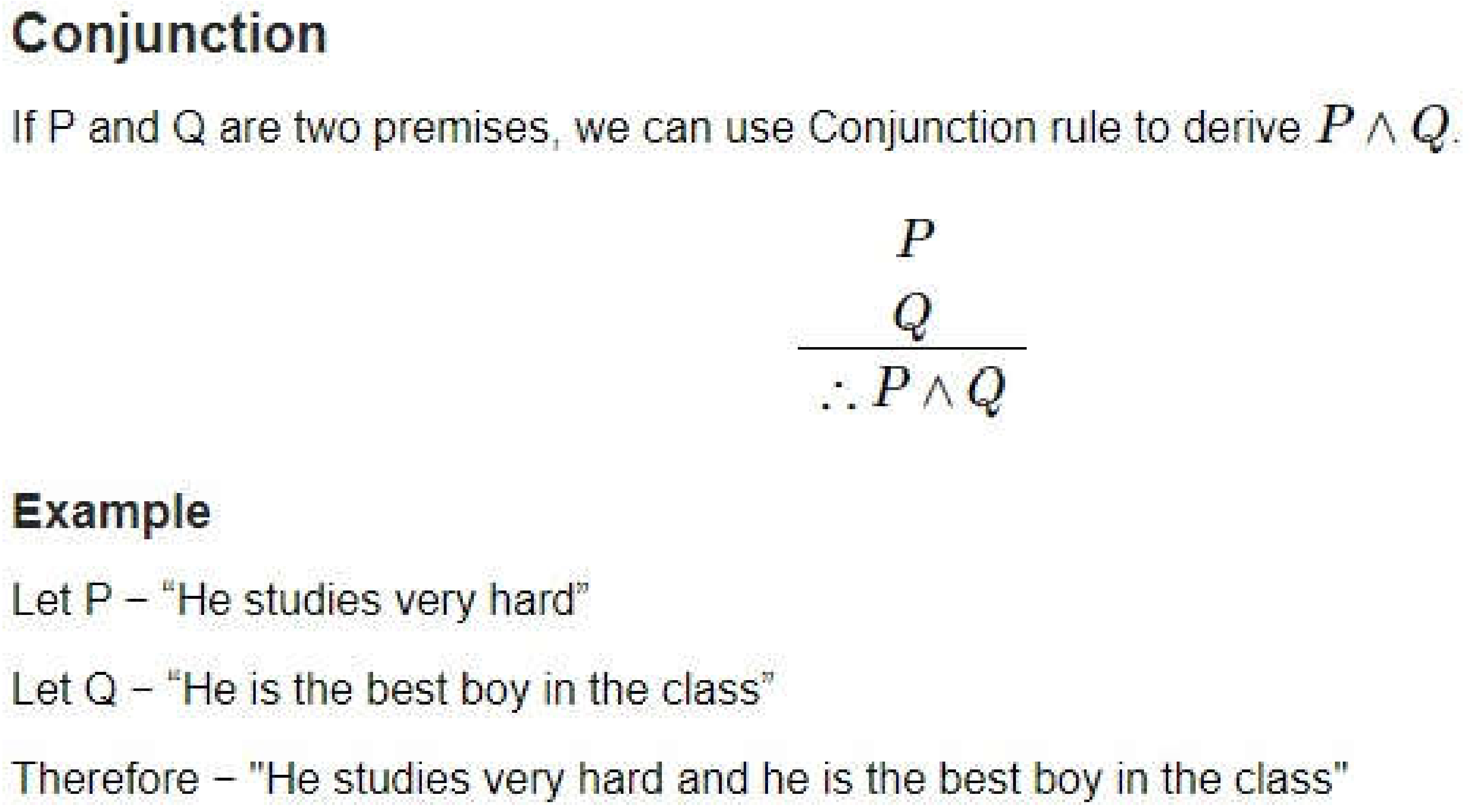
$$\frac{P}{\therefore P \lor Q}$$

Example

Let P be the proposition, “He studies very hard” is true

Therefore − "Either he studies very hard Or he is a very bad student." Here Q is the proposition “he is a very bad student”.

Inference rule & theorem proving



Conjunction

If P and Q are two premises, we can use Conjunction rule to derive $P \wedge Q$.

$$\begin{array}{c} P \\ Q \\ \hline \therefore P \wedge Q \end{array}$$

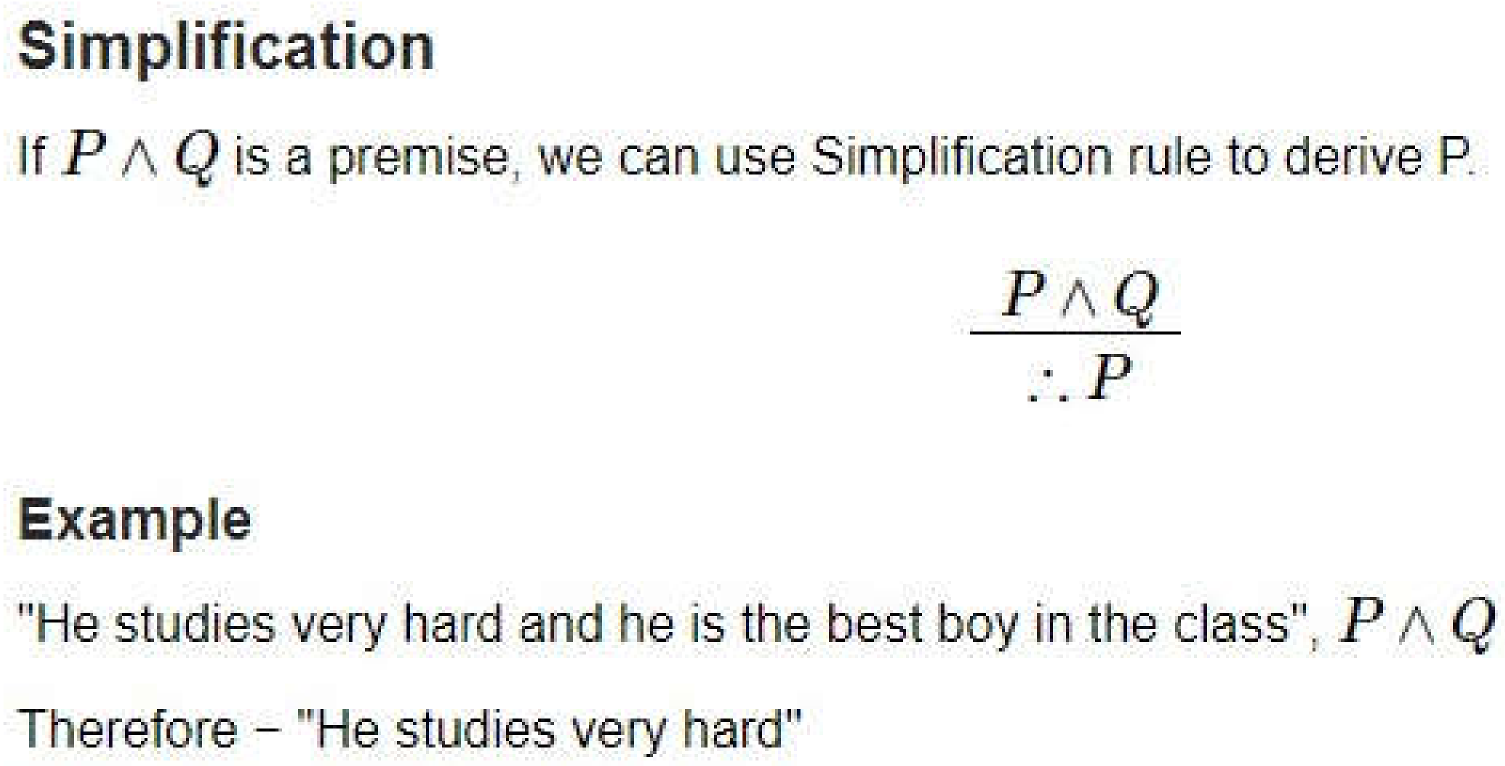
Example

Let P − "He studies very hard"

Let Q − "He is the best boy in the class"

Therefore − "He studies very hard and he is the best boy in the class"

Inference rule & theorem proving



Simplification

If $P \land Q$ is a premise, we can use Simplification rule to derive P.

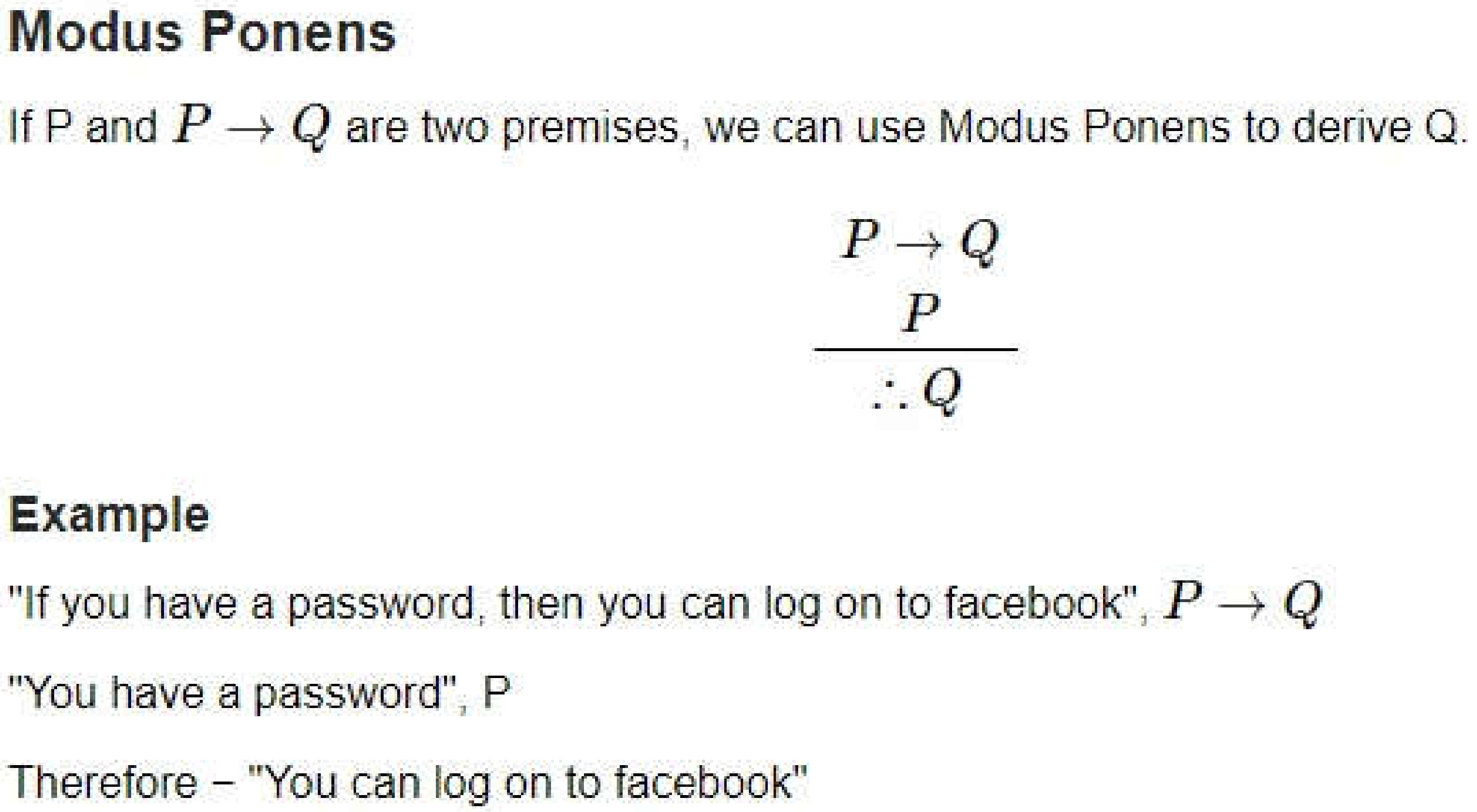
$$\frac{P \land Q}{\therefore P}$$

Example

"He studies very hard and he is the best boy in the class", $P \land Q$

Therefore − "He studies very hard"

Inference rule & theorem proving



Modus Ponens

If P and $P \rightarrow Q$ are two premises, we can use Modus Ponens to derive Q.

$$\frac{\begin{array}{c} P \rightarrow Q \\ P \end{array}}{\therefore Q}$$

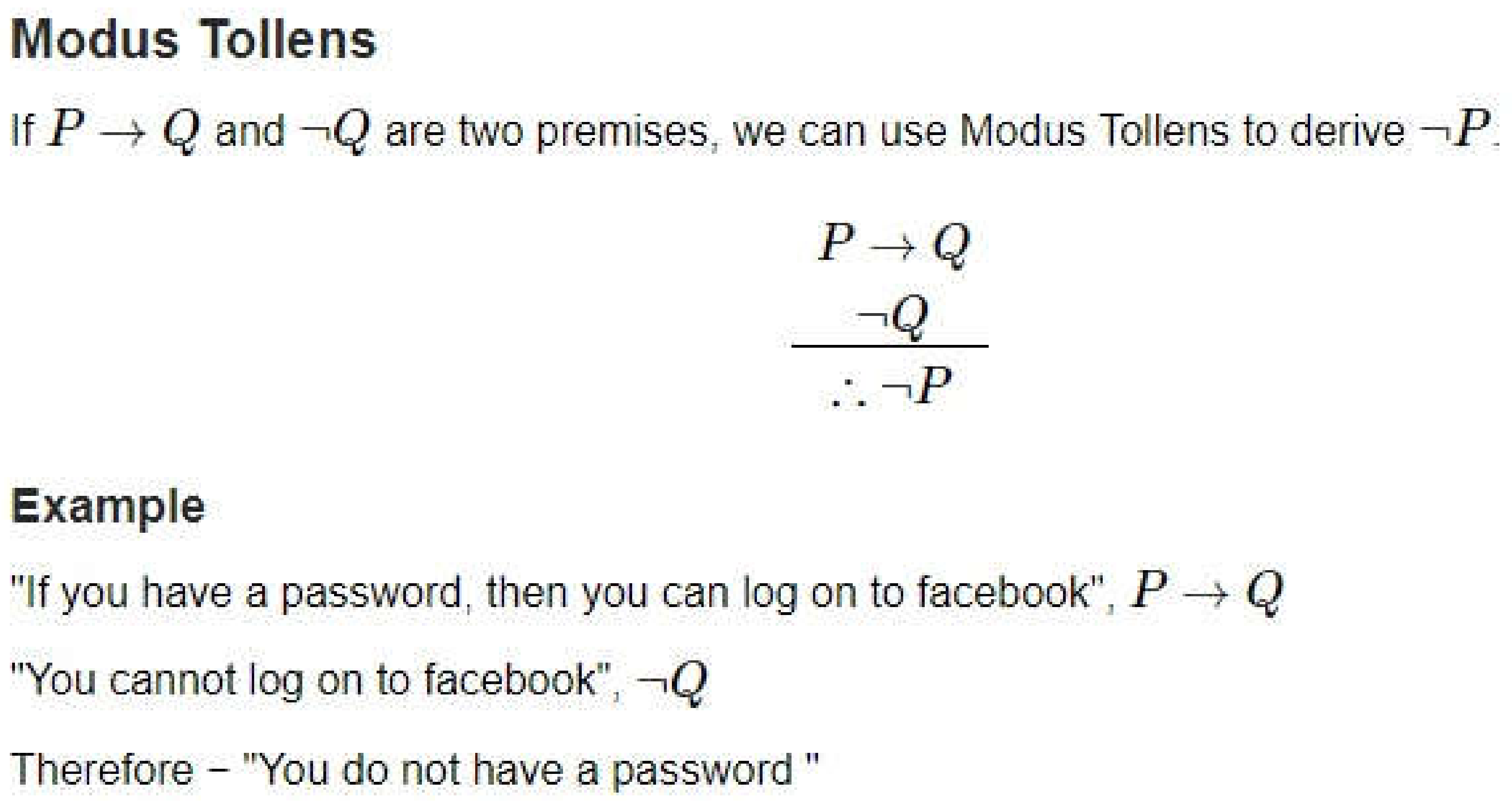
Example

"If you have a password, then you can log on to facebook", $P \rightarrow Q$

"You have a password", P

Therefore – "You can log on to facebook"

Inference rule & theorem proving



Modus Tollens

If $P \to Q$ and $\neg Q$ are two premises, we can use Modus Tollens to derive $\neg P$.

$$\begin{array}{c} P \to Q \\ \neg Q \\ \hline \therefore \neg P \end{array}$$

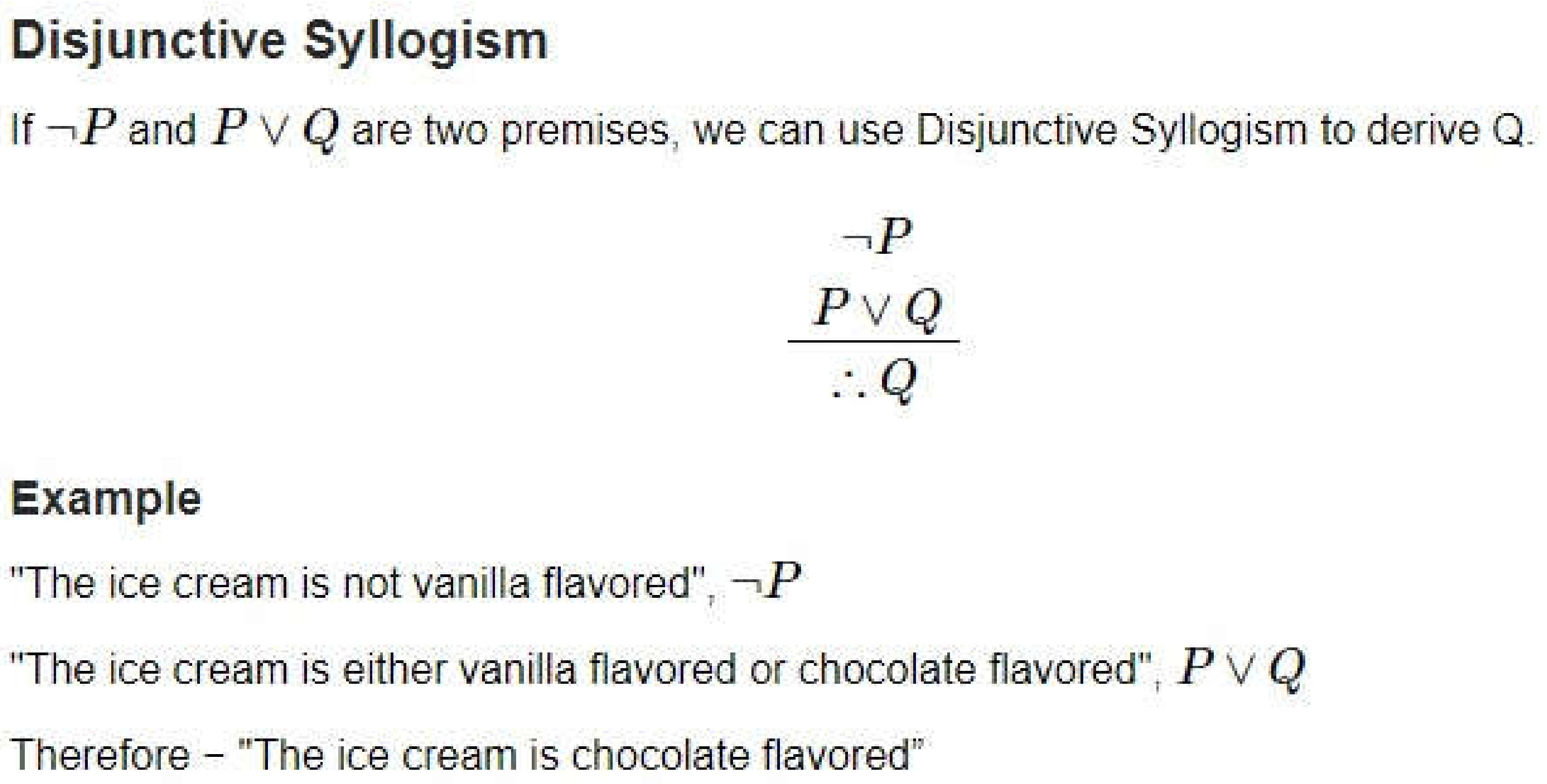
Example

"If you have a password, then you can log on to facebook", $P \to Q$

"You cannot log on to facebook", $\neg Q$

Therefore – "You do not have a password "

Inference rule & theorem proving



Disjunctive Syllogism

If $\neg P$ and $P \vee Q$ are two premises, we can use Disjunctive Syllogism to derive Q.

$$\frac{\begin{array}{c}\neg P \\ P \vee Q\end{array}}{\therefore Q}$$

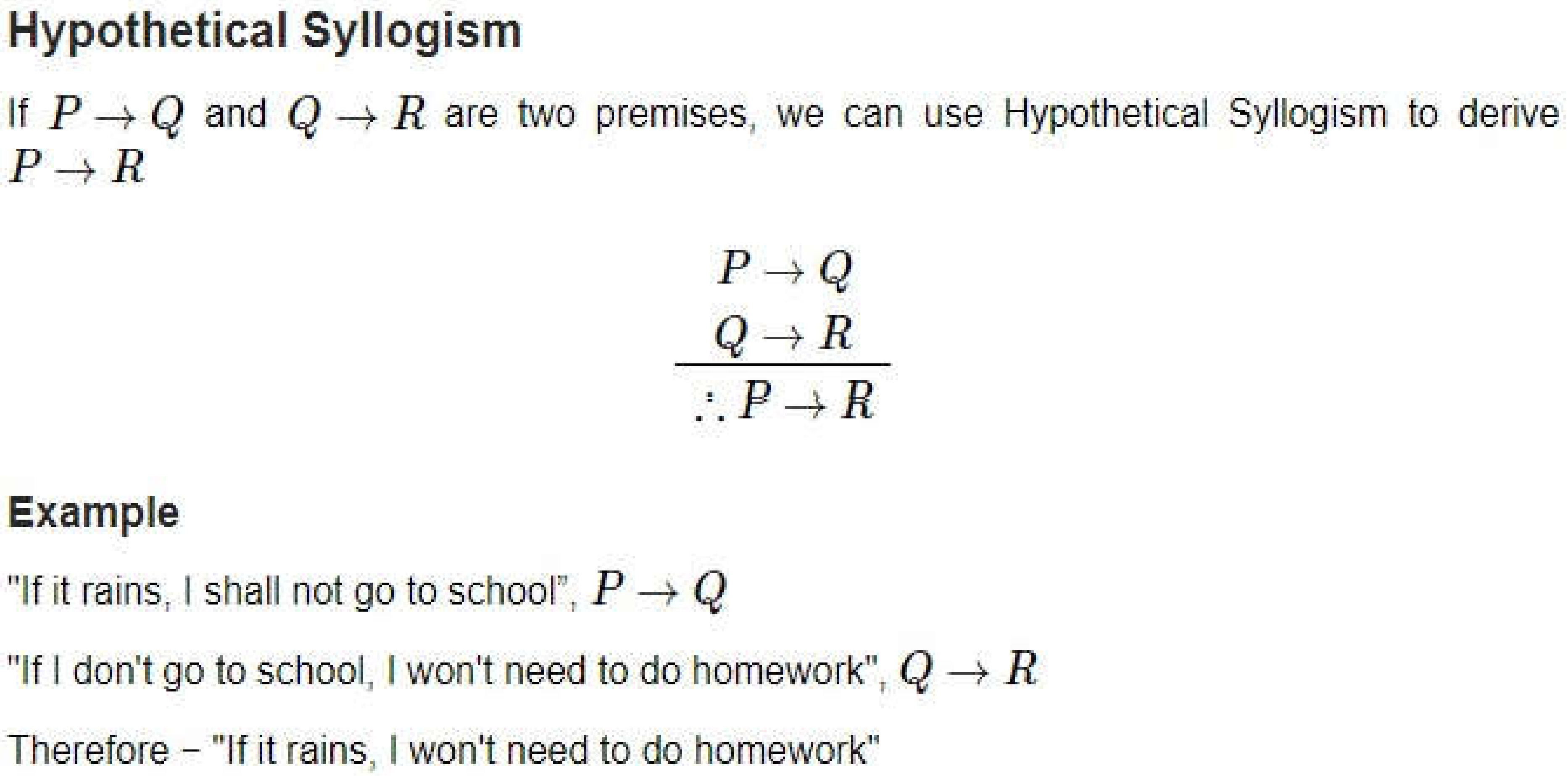
Example

"The ice cream is not vanilla flavored", $\neg P$

"The ice cream is either vanilla flavored or chocolate flavored", $P \vee Q$

Therefore − "The ice cream is chocolate flavored"

Inference rule & theorem proving



Hypothetical Syllogism

If $P \rightarrow Q$ and $Q \rightarrow R$ are two premises, we can use Hypothetical Syllogism to derive $P \rightarrow R$

$$\begin{array}{c} P \rightarrow Q \\ Q \rightarrow R \\ \hline \therefore P \rightarrow R \end{array}$$

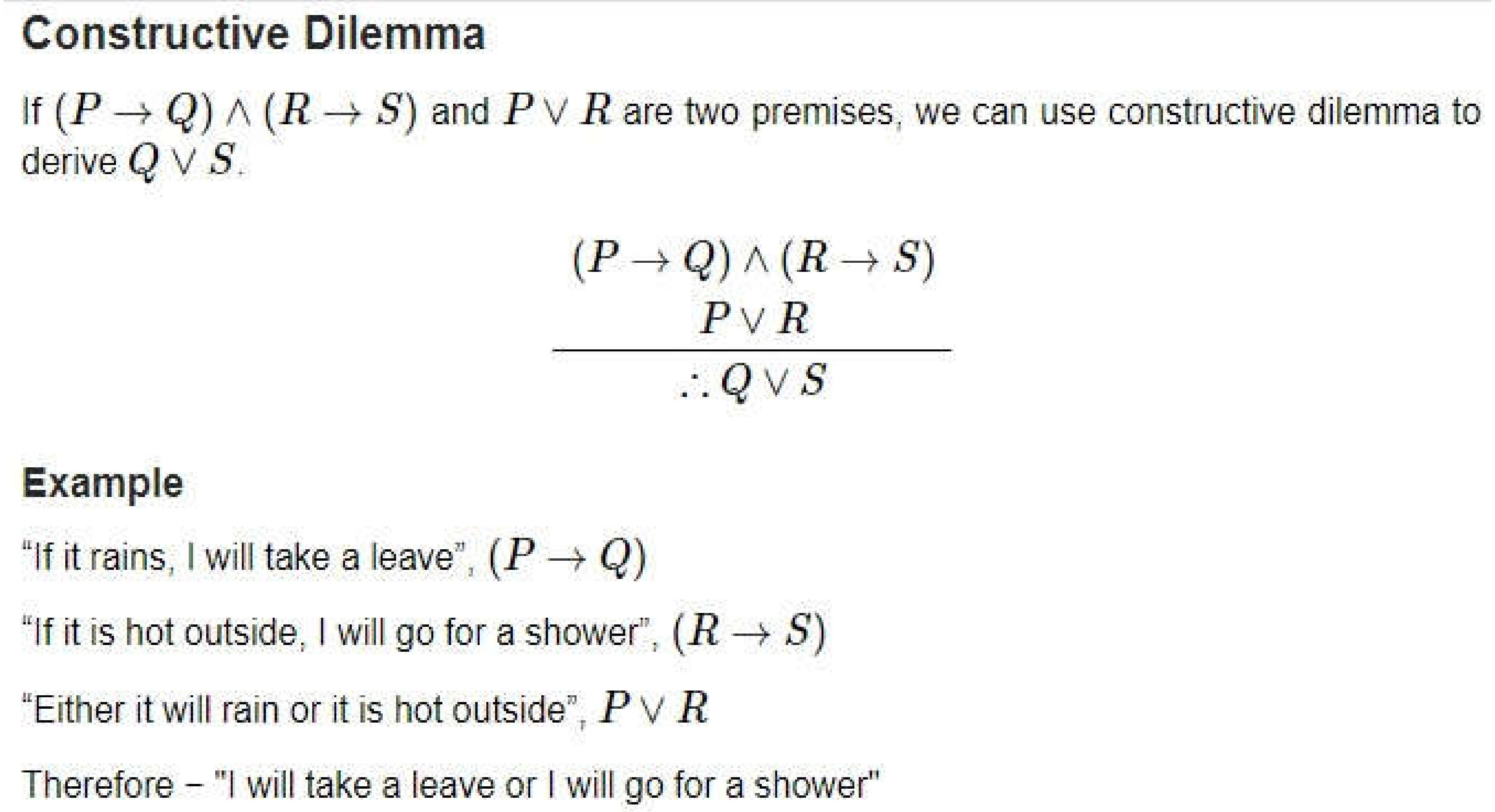
Example

"If it rains, I shall not go to school", $P \rightarrow Q$

"If I don't go to school, I won't need to do homework", $Q \rightarrow R$

Therefore − "If it rains, I won't need to do homework"

Inference rule & theorem proving



Constructive Dilemma

If $(P \rightarrow Q) \land (R \rightarrow S)$ and $P \lor R$ are two premises, we can use constructive dilemma to derive $Q \lor S$.

$$\frac{\begin{matrix}(P \rightarrow Q) \land (R \rightarrow S) \\ P \lor R\end{matrix}}{\therefore Q \lor S}$$

Example

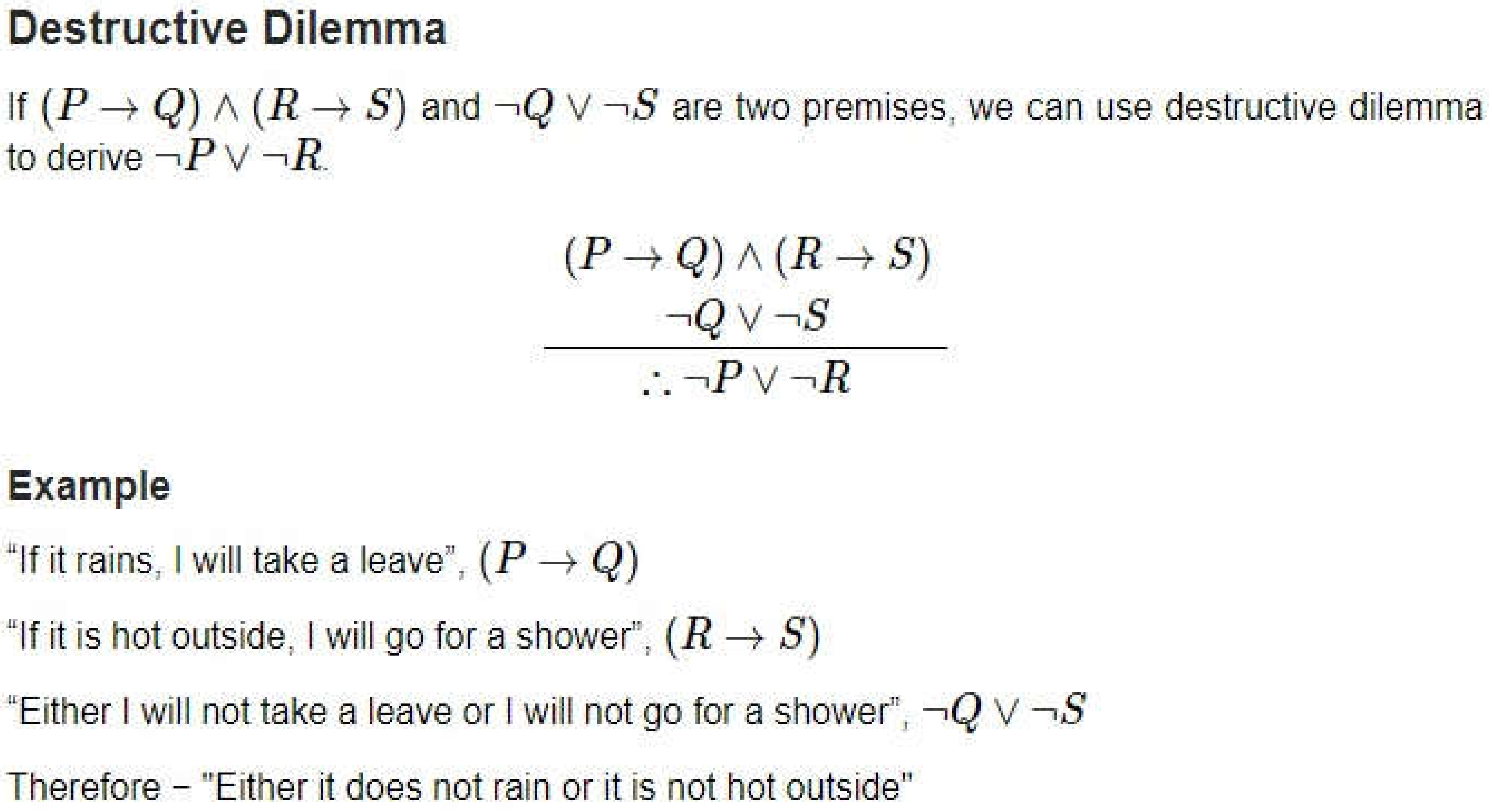
"If it rains, I will take a leave", $(P \rightarrow Q)$

"If it is hot outside, I will go for a shower", $(R \rightarrow S)$

"Either it will rain or it is hot outside", $P \lor R$

Therefore − "I will take a leave or I will go for a shower"

Inference rule & theorem proving



Destructive Dilemma

If $(P \rightarrow Q) \land (R \rightarrow S)$ and $\neg Q \lor \neg S$ are two premises, we can use destructive dilemma to derive $\neg P \lor \neg R$.

$$\frac{\begin{array}{c}(P \rightarrow Q) \land (R \rightarrow S)\\ \neg Q \lor \neg S\end{array}}{\therefore \neg P \lor \neg R}$$

Example

"If it rains, I will take a leave", $(P \rightarrow Q)$

"If it is hot outside, I will go for a shower", $(R \rightarrow S)$

"Either I will not take a leave or I will not go for a shower", $\neg Q \lor \neg S$

Therefore − "Either it does not rain or it is not hot outside"

Forward Chaining

Forward chaining is a form of reasoning which start with atomic sentences in the knowledge base and applies inference rules (Modus Ponens) in the forward direction to extract more data until a goal is reached.

The Forward-chaining algorithm starts from known facts, triggers all rules whose premises are satisfied, and add their conclusion to the known facts. This process repeats until the problem is solved.

Forward Chaining: Example

"As per the law, it is a crime for an American to sell weapons to hostile nations. Country A, an enemy of America, has some missiles, and all the missiles were sold to it by Robert, who is an American citizen.“

Prove that "Robert is criminal.“

To solve the above problem, first, we will convert all the above facts into first-order definite clauses, and then we will use a forward-chaining algorithm to reach the goal.

Forward Chaining: Example

Facts Conversion into FOL:

It is a crime for an American to sell weapons to hostile nations. (Let's say p, q, and r are variables)

American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p)   ...(1)

Country A has some missiles. ?p Owns(A, p) ∧ Missile(p). It can be written in two definite clauses by using Existential Instantiation, introducing new Constant T1.

Owns(A, T1)   ......(2)
Missile(T1)   ....(3)

Forward Chaining: Example

Facts Conversion into FOL:

All of the missiles were sold to country A by Robert.

?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p,A)   ......(4)

Missiles are weapons.

Missile(p) → Weapons (p)   ....(5)

Enemy of America is known as hostile.

Enemy(p, America) →Hostile(p)   ....(6)

Country A is an enemy of America.

Enemy (A, America)   ....(7)

Robert is American

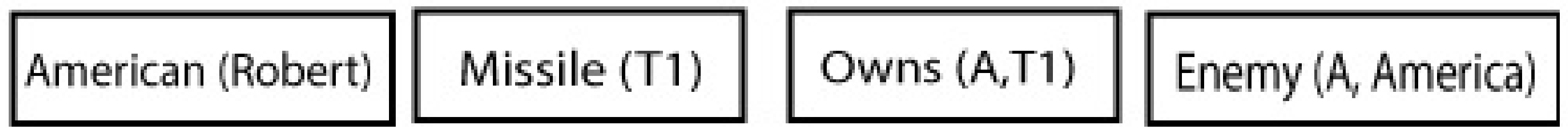
American(Robert).   ....(8)

Forward Chaining: Example

Step-1:

In the first step we will start with the known facts and will choose the sentences which do not have implications, such as: American(Robert), Enemy(A, America), Owns(A, T1), and Missile(T1).

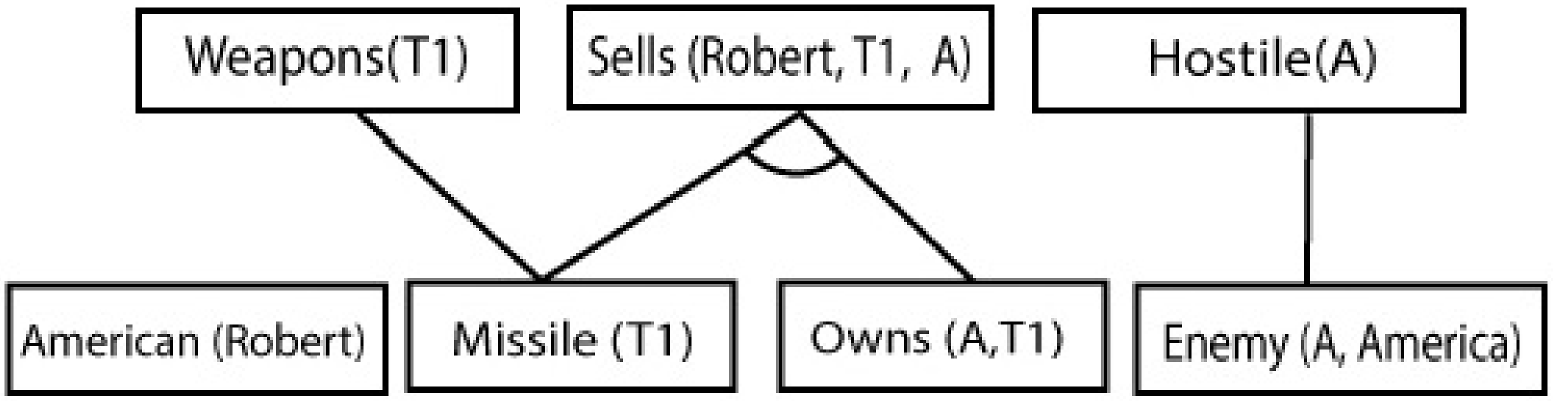
All these facts will be represented as below:



Forward Chaining: Example

Step-2:

Rule-(6) is satisfied with the substitution(p/A), so Hostile(A) is added and which infers from Rule-(7).

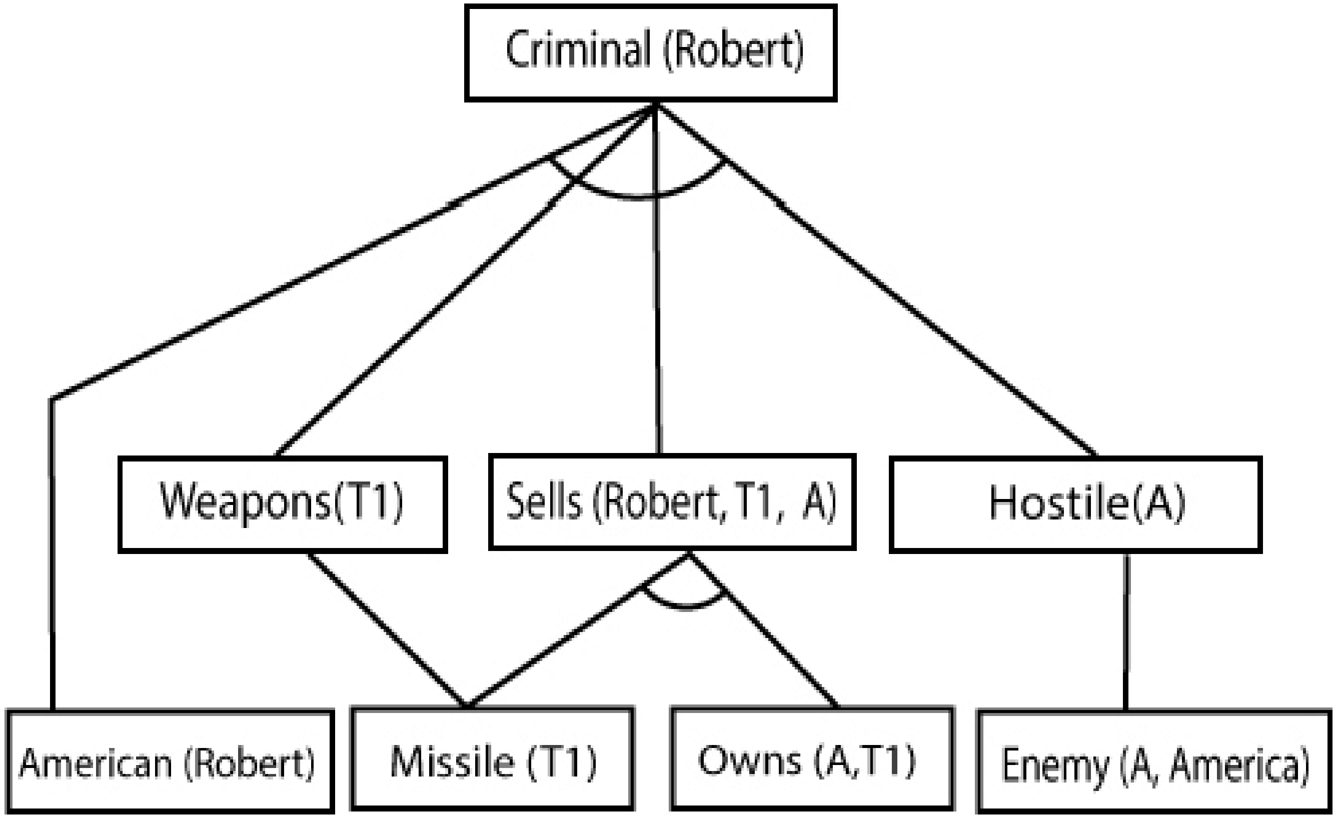


Forward Chaining: Example

Step-3:

At step-3, as we can check Rule-(1) is satisfied with the substitution {p/Robert, q/T1, r/A}.

so we can add Criminal(Robert) which infers all the available facts. And hence we reached our goal statement.



Backward Chaining

A backward chaining algorithm is a form of reasoning, which starts with the goal and works backward, chaining through rules to find known facts that support the goal.

Example:

In backward-chaining, we will use the same above example, and will rewrite all the rules:

American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1)

Owns(A, T1) ....(2)

Backward Chaining: Example

Missile(T1)

?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) ....(4)

Missile(p) → Weapons (p) ....(5)

Enemy(p, America) →Hostile(p) ....(6)

Enemy (A, America) ....(7)

American(Robert). ....(8)

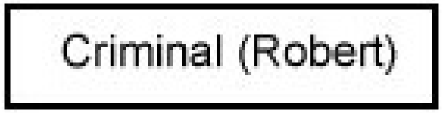
Backward Chaining: Example

In Backward chaining, we will start with our goal predicate, which is Criminal(Robert), and then infer further rules.

Step-1:

At the first step, we will take the goal fact. And from the goal fact, we will infer other facts, and at last, we will prove those facts true.

So our goal fact is "Robert is Criminal," so following is the predicate of it.

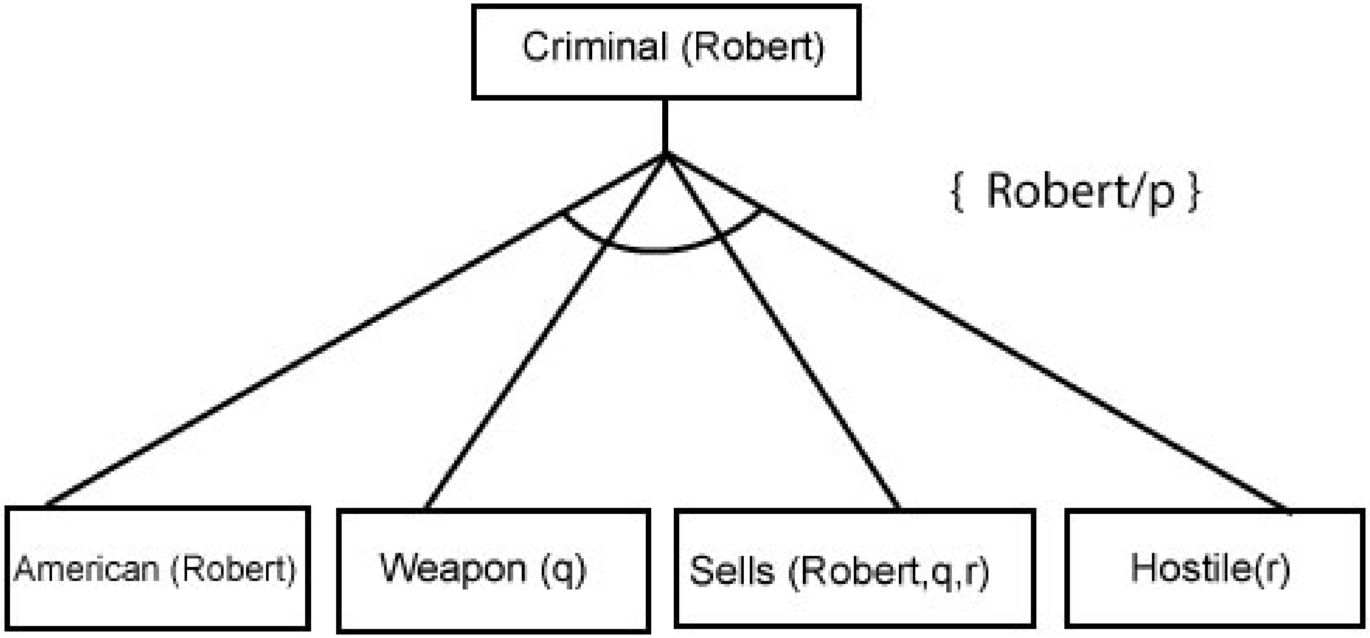


Backward Chaining: Example

Step-2:

We will infer other facts form goal fact which satisfies the rules. So as we can see in Rule-1, the goal predicate Criminal (Robert) is present with substitution {Robert/P}.

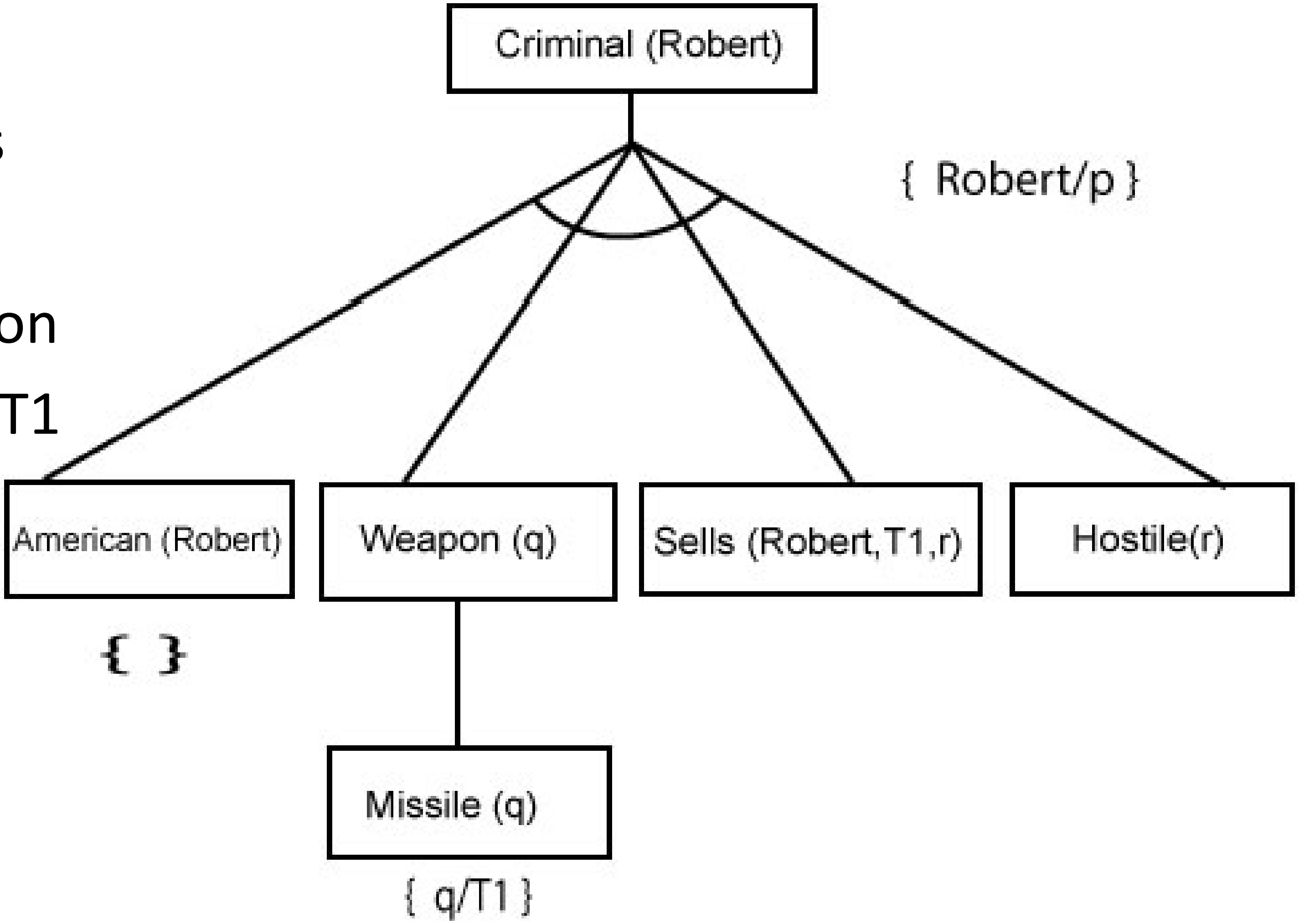
So we will add all the conjunctive facts below the first level and will replace p with Robert.



Backward Chaining: Example

Step-3: We will extract further fact Missile(q) which infer from Weapon(q), as it satisfies Rule-(5).

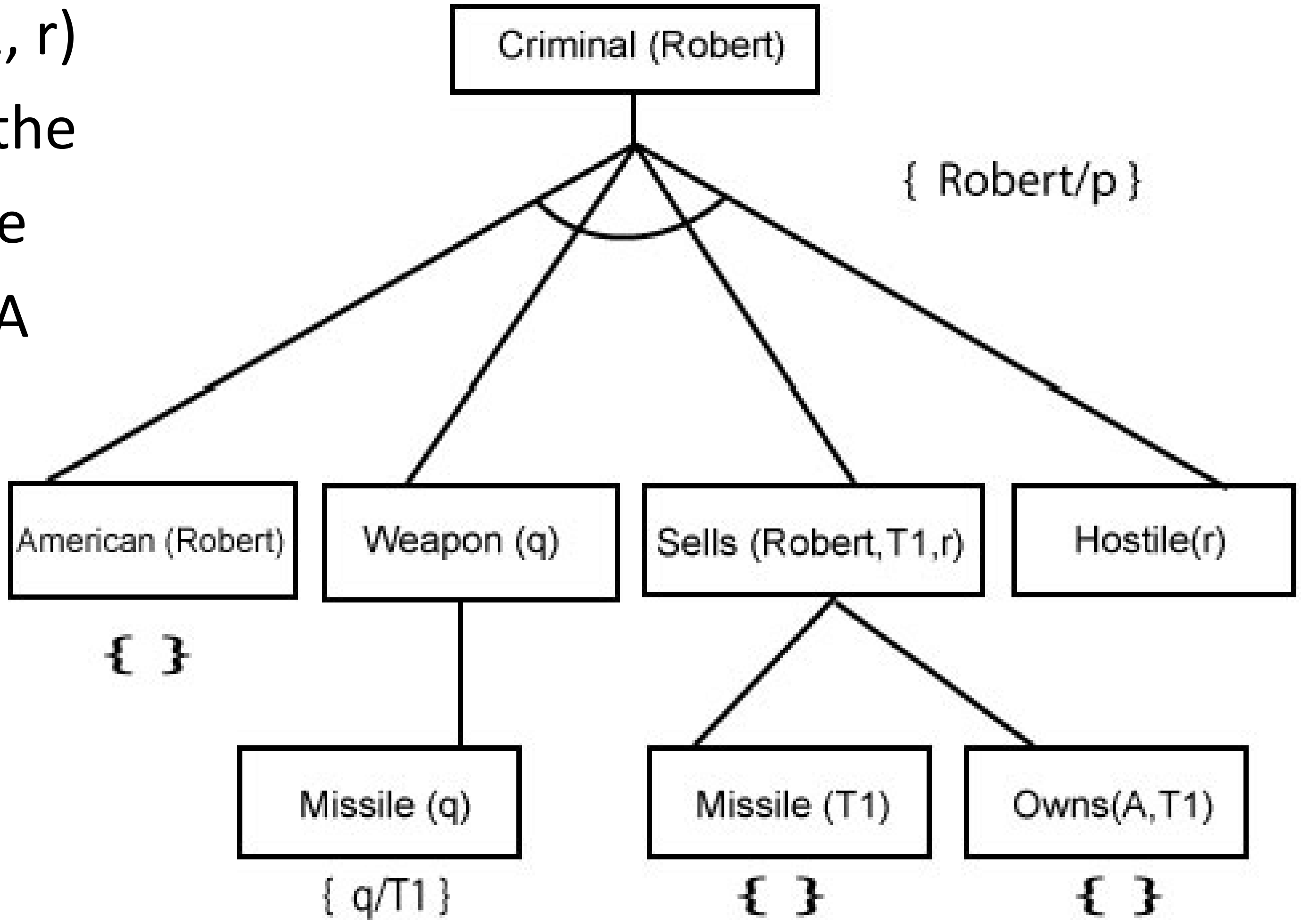
Weapon (q) is also true with the substitution of a constant T1 at q.



Backward Chaining: Example

Step-4: we can infer facts Missile(T1) and Owns(A, T1) form Sells(Robert, T1, r) which satisfies the Rule- 4, with the substitution of A in place of r.

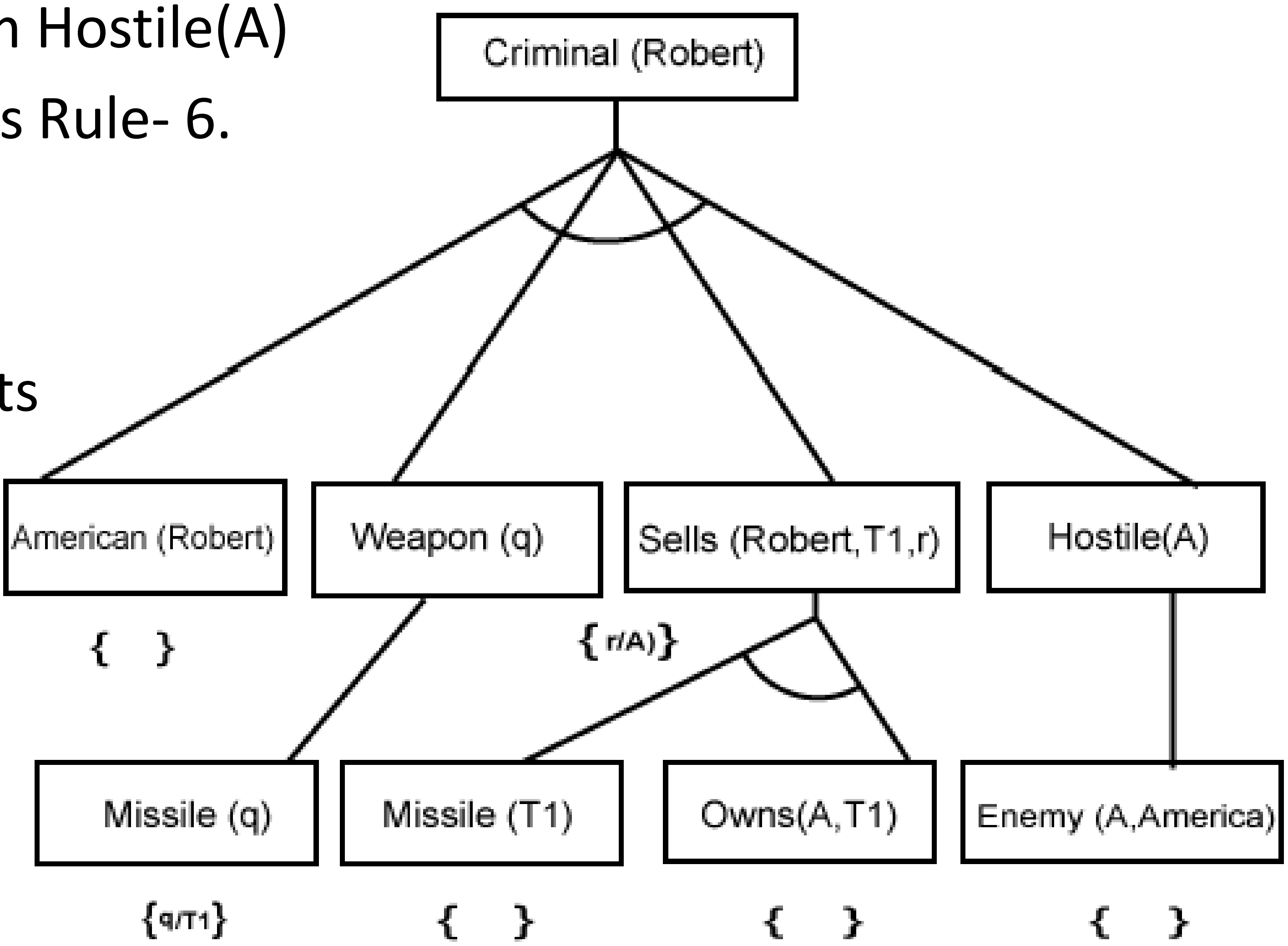
So these two statements are proved here.



Backward Chaining: Example

At **step-5,** we can infer the fact Enemy(A, America) from Hostile(A) which satisfies Rule- 6.

And hence all the statements are proved true using backward chaining.



Forward Chaining: Example 2

Axioms:

1. If D barks and D eats bone, then D is a dog.
2. If V is cold and V is sweet, then V is ice-cream.
3. If D is a dog, then D is black.
4. If V is ice-cream, then it is Vanilla.

Derive forward chaining using the below known facts to prove Tomy is black.

- Tomy barks.
- Tomy eats bone.

Forward Chaining: Example 2

Solution:

- Given Tomy barks.

- From (1), it is clear:

  If Tomy barks and Tomy eats bone, then Tomy is a dog.

- From (3), it is clear:

  If Tomy is a dog, then Tomy is black.

- Hence, it is proved that Tomy is black.

Forward Chaining: Example 3

Consider the below axioms:

1. Gita loves all types of clothes.
2. Suits are clothes.
3. Jackets are clothes.
4. Anything any wear and isn’t bad is clothes.
5. Sita wears skirt and is good.
6. Renu wears anything Sita wears.

Apply backward chaining and prove that Gita loves Kurtis.

Forward Chaining: Example 3

Convert the given axioms into FOPL as:

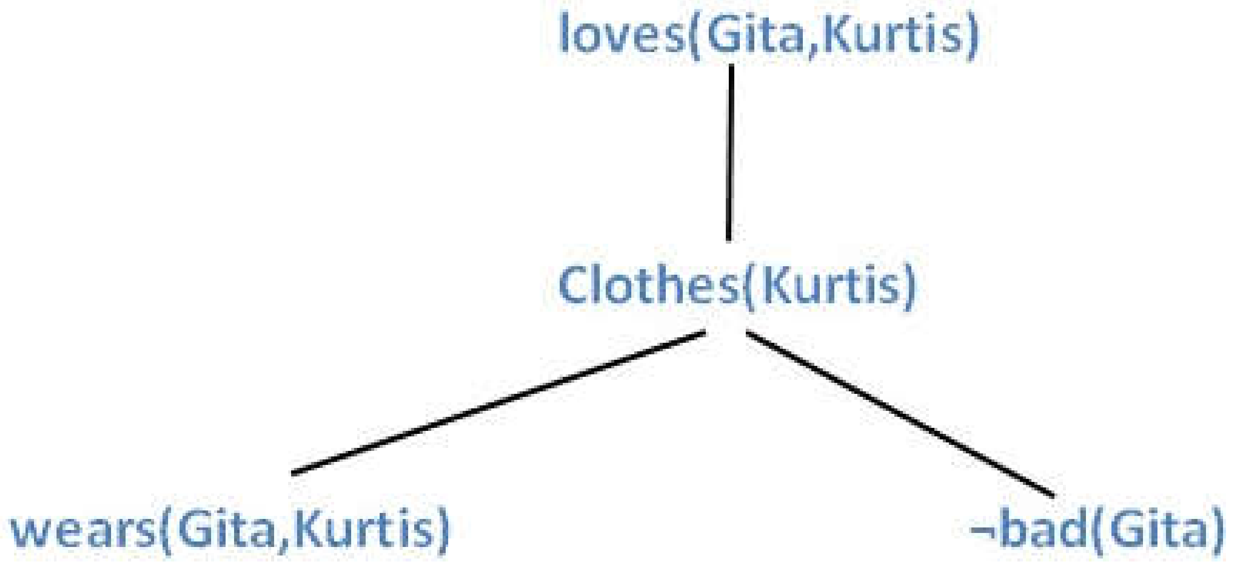
1. x: clothes(x)→loves(Gita, x).
2. Suits(x)→Clothes(x).
3. Jackets(x)→Clothes(x).
4. wears(x,y)Λ ¬bad(y)→Clothes(x)
5. wears(Sita,skirt) Λ ¬good(Sita)
6. wears(Sita,x)→wears(Renu,x)

Forward Chaining: Example 3

To prove: Gita loves Kurtis.

First Order Predicate Logic: loves(Gita, Kurtis).

On applying forward chaining in the below graph:



Thus, it is proved that Gita loves Kurtis.

Backward Chaining: Example 2

Consider the previous example 2 from forward chaining:

Axioms:

1. If D barks and D eats bone, then D is a dog.
2. If V is cold and V is sweet, then V is ice-cream.
3. If D is a dog, then D is black.
4. If V is ice-cream, then it is Vanilla.

Derive backward chaining using the given known facts to prove Tomy is black.

- Tomy barks.
- Tomy eats bone.

Backward Chaining: Example 2

Solution:

- On replacing D with Tomy in (3), it becomes:

  If Tomy is a dog, then Tomy is black.

  Thus, the goal is matched with the above axiom.

- Now, we have to prove Tomy is a dog. …*(new goal)*

  Replace D with Tomy in (1), it will become:

  If Tomy barks and Tomy eats bone, then Tomy is a dog. …*(new goal)*

  Again, the goal is achieved.

Backward Chaining: Example 2

Solution:

- Now, we have to prove that Tomy barks and Tomy eats bone. *…(new goal)*

  As we can see, the goal is a combination of two sentences which can be further divided as:

  Tomy barks.

  Tomy eats bone.

From (1), it is clear that Tomy is a dog.

Hence, Tomy is black.

Note: Statement (2) and (4) are not used in proving the given axiom. So, it is clear that goal never matches the negated versions of the axioms. Always Modus Ponen is used, rather Modus Tollen.

Backward Chaining: Example 3

Consider the below axioms:

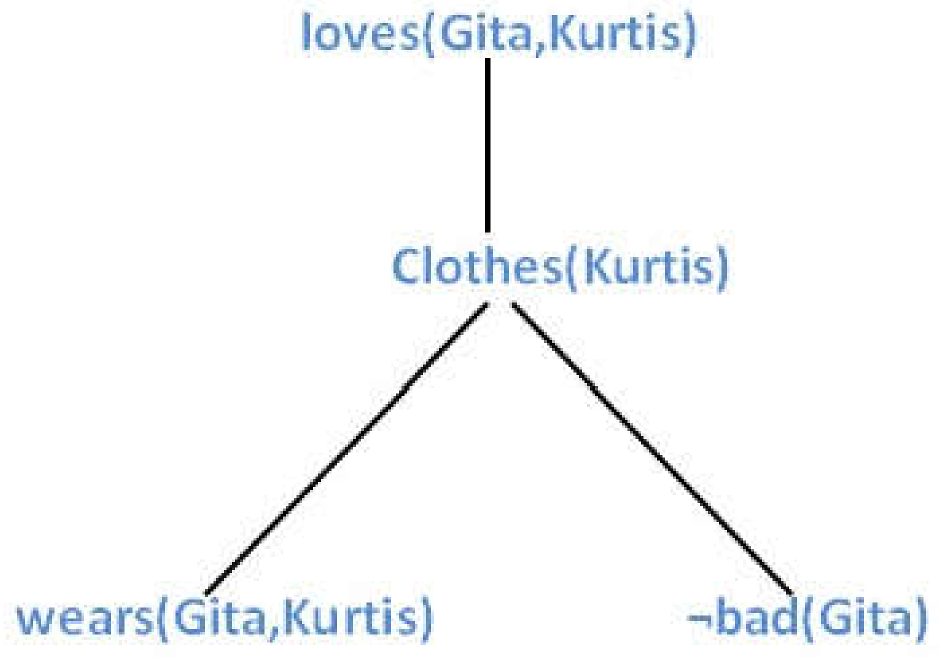
1. x: clothes(x)→loves(Gita, x).
2. Suits(x)→Clothes(x).
3. Jackets(x)→Clothes(x).
4. wears(x,y)→Λ ¬bad(y)→Clothes(x)
5. wears(Sita,skirt)Λ ¬good(Sita)
6. wears(Sita,x)→wears(Renu,x)

To prove: Gita loves Kurtis.

FOPL: loves(Gita, Kurtis).

Backward Chaining: Example 3

Apply backward chaining in the below graph:



It is clear from the above graph Gita wears Kurtis and does not look bad. Hence, Gita loves Kurtis.

Resolution

Resolution method is an inference rule which is used in both Propositional as well as First-order Predicate Logic in different ways.

This method is basically used for proving the satisfiability of a sentence.

The key idea for the resolution method is to use the knowledge base and negated goal to obtain null clause(which indicates contradiction).

Resolution method is also called Proof by Refutation.

Resolution: in propositional calculus

In its simplest form Resolution is the inference rule:

```
{A OR C, B OR (NOT C)}
----------------------
       A OR B
```

More in general the Resolution Inference Rule is:

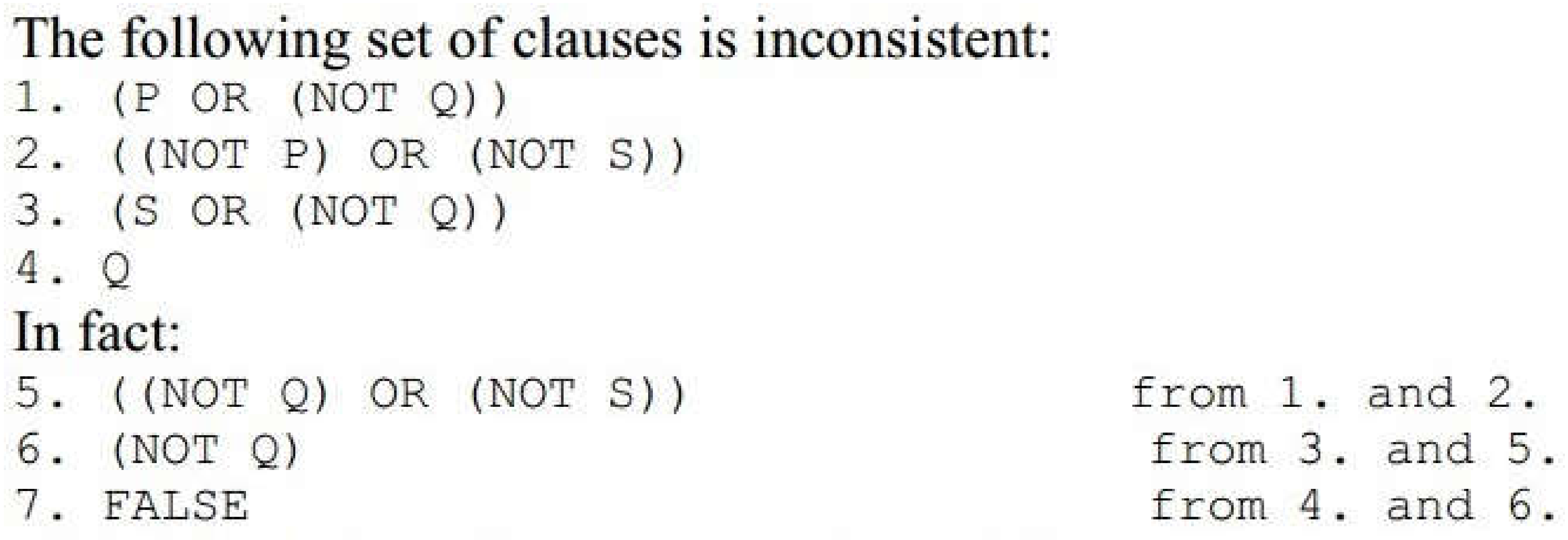
Given as premises the clauses C1 and C2, where C1 contains the literal L and C2 contains the literal (NOT L), infer the clause C, called the Resolvent of C1 and C2, where C is the union of (C1 - {L}) and (C2 -{(NOT L)})

In symbols:

```
                 {C1, C2}
    ----------------------------------
    (C1 - {L}) UNION (C2 - {(NOT L)})
```

Resolution: in propositional calculus

Example:



The following set of clauses is inconsistent:

```
1. (P OR (NOT Q))
2. ((NOT P) OR (NOT S))
3. (S OR (NOT Q))
4. Q
```

In fact:

```
5. ((NOT Q) OR (NOT S))                       from 1. and 2.
6. (NOT Q)                                     from 3. and 5.
7. FALSE                                       from 4. and 6.
```

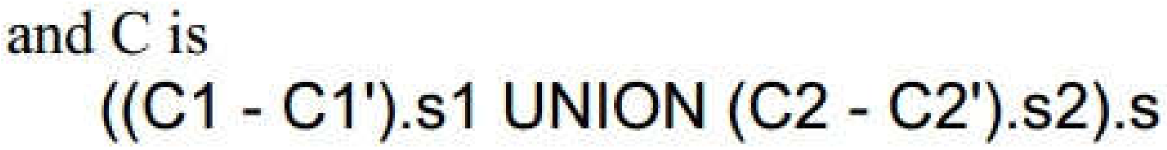
Resolution: in first order logic

Given clauses C1 and C2, a clause C is a RESOLVENT of C1 and C2, if:

1. There is a subset C1' = {A1, .., Am} of C1 of literals of the same sign, say positive, and a subset C2' = {B1, .., Bn} of C2 of literals of the opposite sign, say negative,
2. There are substitutions s1 and s2 that replace variables in C1' and C2' so as to have new variables,
3. C2'' is obtained from C2 removing the negative signs from B1 .. Bn
4. There is an Most General Unifier s for the union of

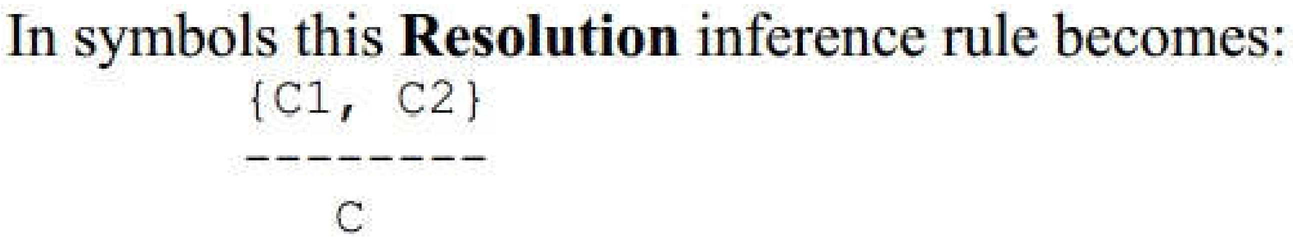
   C1'.s1 and C2''.s2

Resolution: in first order logic



and C is

((C1 - C1').s1 UNION (C2 - C2').s2).s

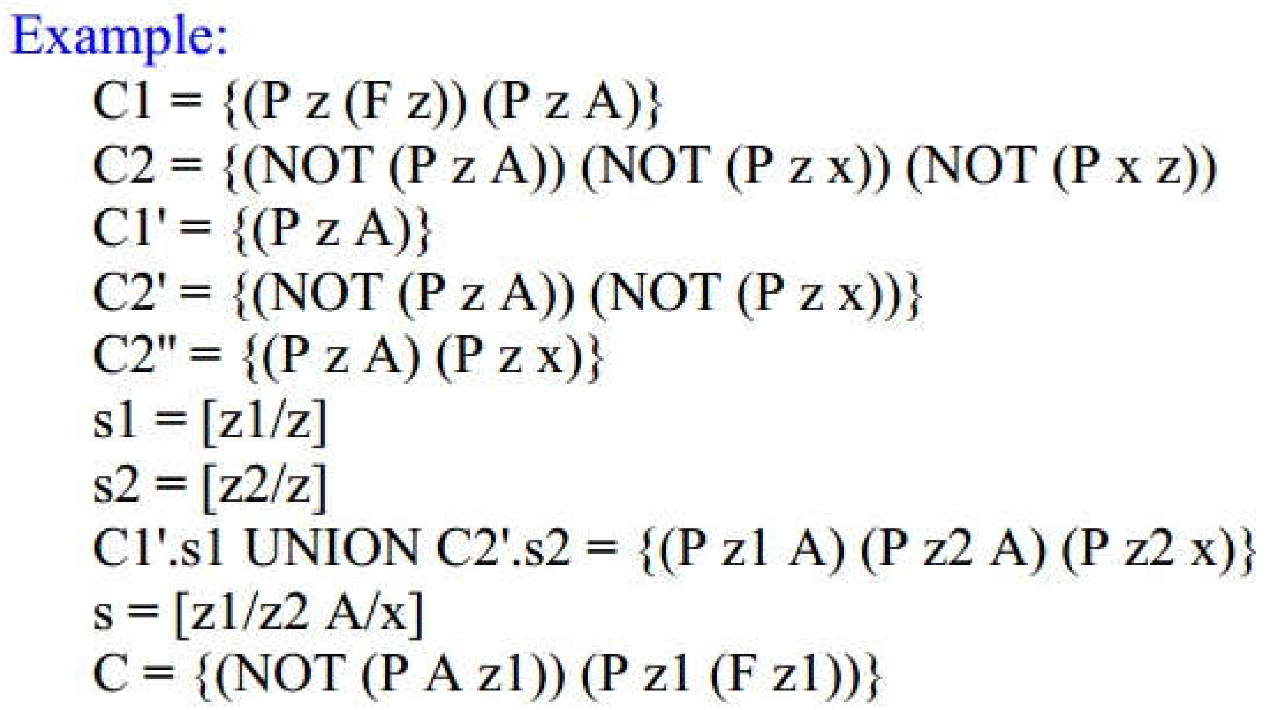


In symbols this **Resolution** inference rule becomes:

```
{C1, C2}
---------
    C
```

If C1' and C2' are singletons (i.e. contain just one literal), the rule is called Binary Resolution.

Resolution: in first order logic



Example:

C1 = {(P z (F z)) (P z A)}
C2 = {(NOT (P z A)) (NOT (P z x)) (NOT (P x z))
C1' = {(P z A)}
C2' = {(NOT (P z A)) (NOT (P z x))}
C2'' = {(P z A) (P z x)}
s1 = [z1/z]
s2 = [z2/z]
C1'.s1 UNION C2'.s2 = {(P z1 A) (P z2 A) (P z2 x)}
s = [z1/z2 A/x]
C = {(NOT (P A z1)) (P z1 (F z1))}

Notice that this application of Resolution has eliminated more than one literal from C2, i.e. it is not a binary resolution.

Resolution

Resolution yields a complete inference algorithm when coupled with any complete search algorithm.

Resolution makes use of the inference rules.

Resolution performs deductive inference.

Resolution uses proof by contradiction.

One can perform Resolution from a Knowledge Base.

Resolution: Principle

$$l_1 \text{ v } \ldots \text{ v } l_k,\ m_1 \text{ v } \ldots \text{ v } m_n$$

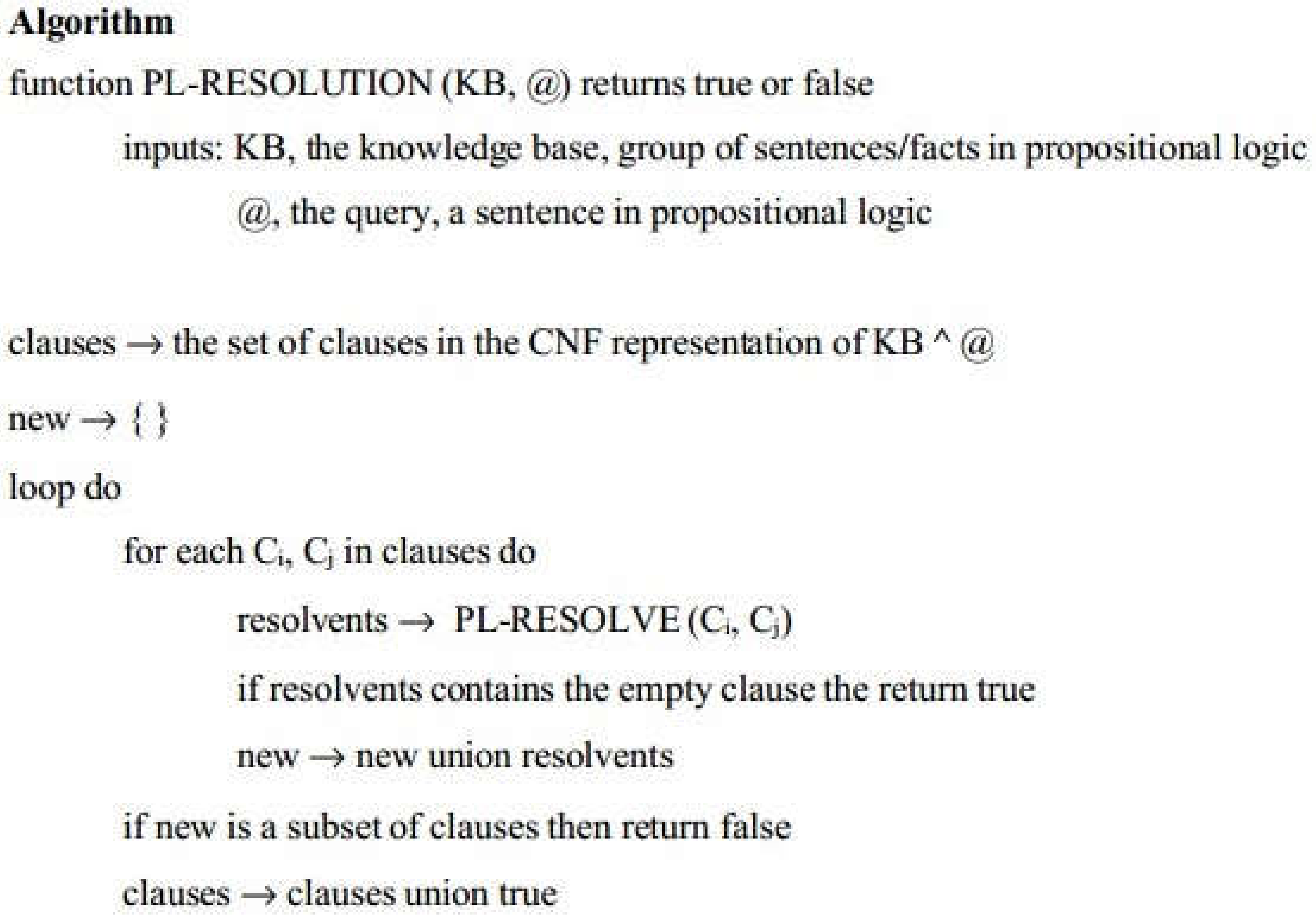
$$\overline{\hspace{20em}}$$

$$l_1\text{v}\ldots\text{v } l_{i-1} \text{ v } l_{i+1} \text{ v}\ldots\text{v } l_k \text{ v } m_1\text{v}\ldots\text{v } m_{j-1}\text{v } m_{j-1}\text{v}\ldots\text{v } m_n$$

li and mj are complementary literals.

Resolution takes two clauses and produces a new clause containing all the literals of the two original clauses except the two complementary literals.

Resolution: Algorithm



Algorithm

```
function PL-RESOLUTION (KB, @) returns true or false
        inputs: KB, the knowledge base, group of sentences/facts in propositional logic
                @, the query, a sentence in propositional logic

clauses → the set of clauses in the CNF representation of KB ^ @
new → { }
loop do
        for each $C_i$, $C_j$ in clauses do
                resolvents →  PL-RESOLVE ($C_i$, $C_j$)
                if resolvents contains the empty clause the return true
                new → new union resolvents
        if new is a subset of clauses then return false
        clauses → clauses union true
```

Resolution: Steps

- Convert the given statements in Predicate/Propositional Logic.
- Convert these statements into Conjunctive Normal Form.
- Negate the Conclusion (Proof by Contradiction).
- Resolve using a Resolution Tree (Unification).

Resolution: Example 1

Now let us see an example which uses resolution.

Problem Statement:

1. Ravi likes all kind of food.
2. Apples and chicken are food
3. Anything anyone eats and is not killed is food
4. Ajay eats peanuts and is still alive
5. Rita eats everything that Ajay eats

Prove by resolution that Ravi likes peanuts using resolution.

Resolution: Example 1

Step 1: Converting the given statements into Predicate/Propositional Logic

i. ∀x : food(x) → likes (Ravi, x)

ii. food (Apple) ^ food (chicken)

iii. ∀a : ∀b: eats (a, b) ^ killed (a) → food (b)

iv. eats (Ajay, Peanuts) ^ alive (Ajay)

v. ∀c : eats (Ajay, c) → eats (Rita, c)

vi. ∀d : alive(d) → ~killed (d)

vii. ∀e: ~killed(e) → alive(e)

Conclusion: likes (Ravi, Peanuts)

Resolution: Example 1

Step 2: Convert into CNF

i. ~food(x) v likes (Ravi, x)

ii. Food (apple)

iii. Food (chicken)

iv. ~ eats (a, b) v killed (a) v food (b)

v. Eats (Ajay, Peanuts)

vi. Alive (Ajay)

vii. ~eats (Ajay, c) V eats (Rita, c)

viii. ~alive (d) v ~ killed (d)

ix. Killed (e) v alive (e)

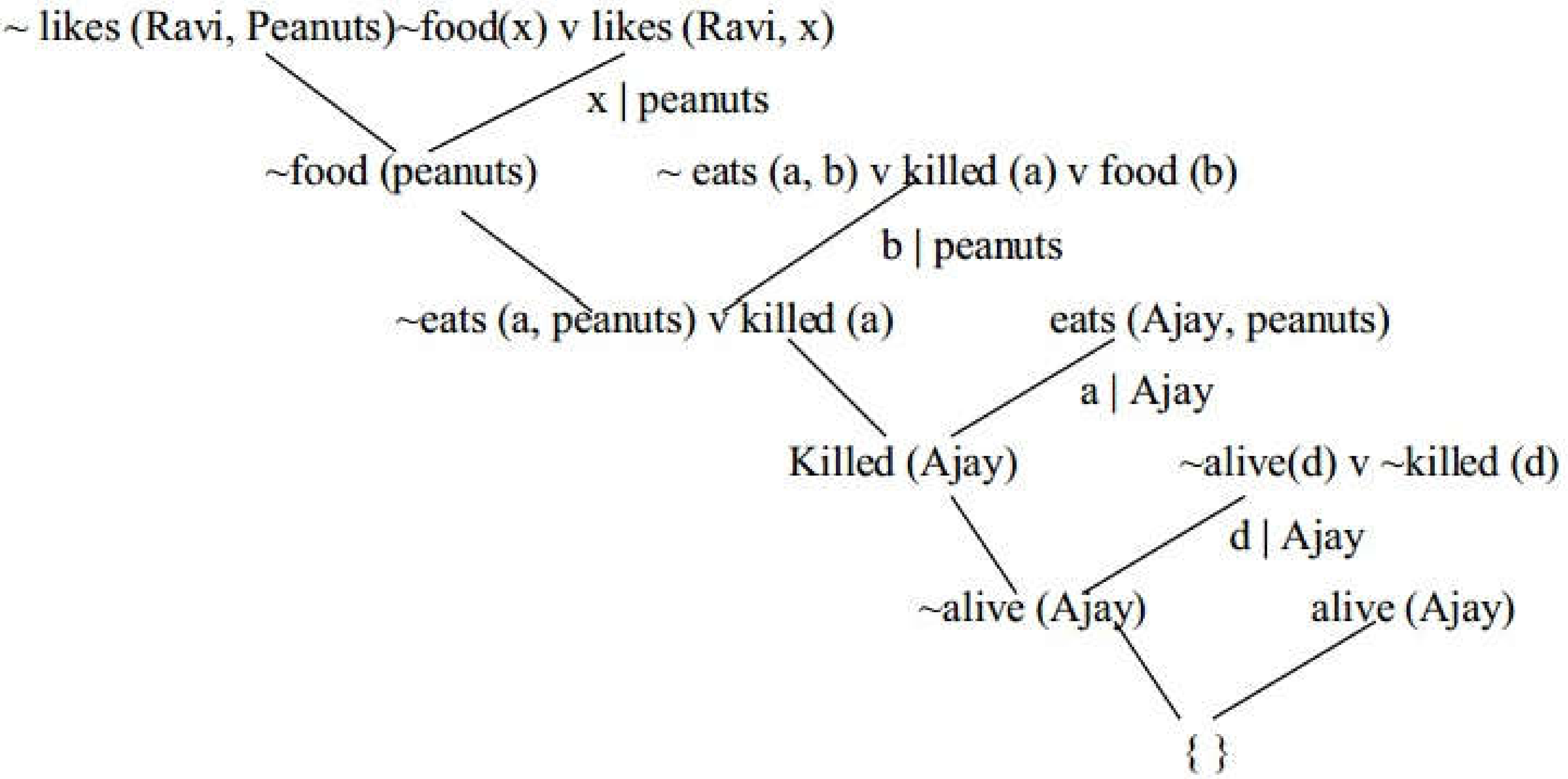
Conclusion: likes (Ravi, Peanuts)

Step 3: Negate the conclusion

~ likes (Ravi, Peanuts)

Resolution: Example 1

Step 4: Resolve using a resolution tree



Resolution: Example 1

Hence we see that the negation of the conclusion has been proved as a complete contradiction with the given set of facts.

Hence the negation is completely invalid or false or the assertion is completely valid or true.

Hence Proved

Resolution: Example 2

Consider the following Knowledge Base:

1. The humidity is high or the sky is cloudy.
2. If the sky is cloudy, then it will rain.
3. If the humidity is high, then it is hot.
4. It is not hot.

Goal: It will rain.

Apply resolution method to prove that the goal is derivable from the given knowledge base.

Resolution: Example 2

Solution:

Let’s construct propositions of the given sentences one by one:

1) Let, P: Humidity is high.

Q: Sky is cloudy.

It will be represented as P V Q.

2) Q: Sky is cloudy. …from(1)

Let, R: It will rain.

It will be represented as bQ $\rightarrow$ R.

Resolution: Example 2

3) P: Humidity is high. …from(1)

Let, S: It is hot.

It will be represented as P $\rightarrow$ S.

4) ¬S: It is not hot.

Applying resolution method:

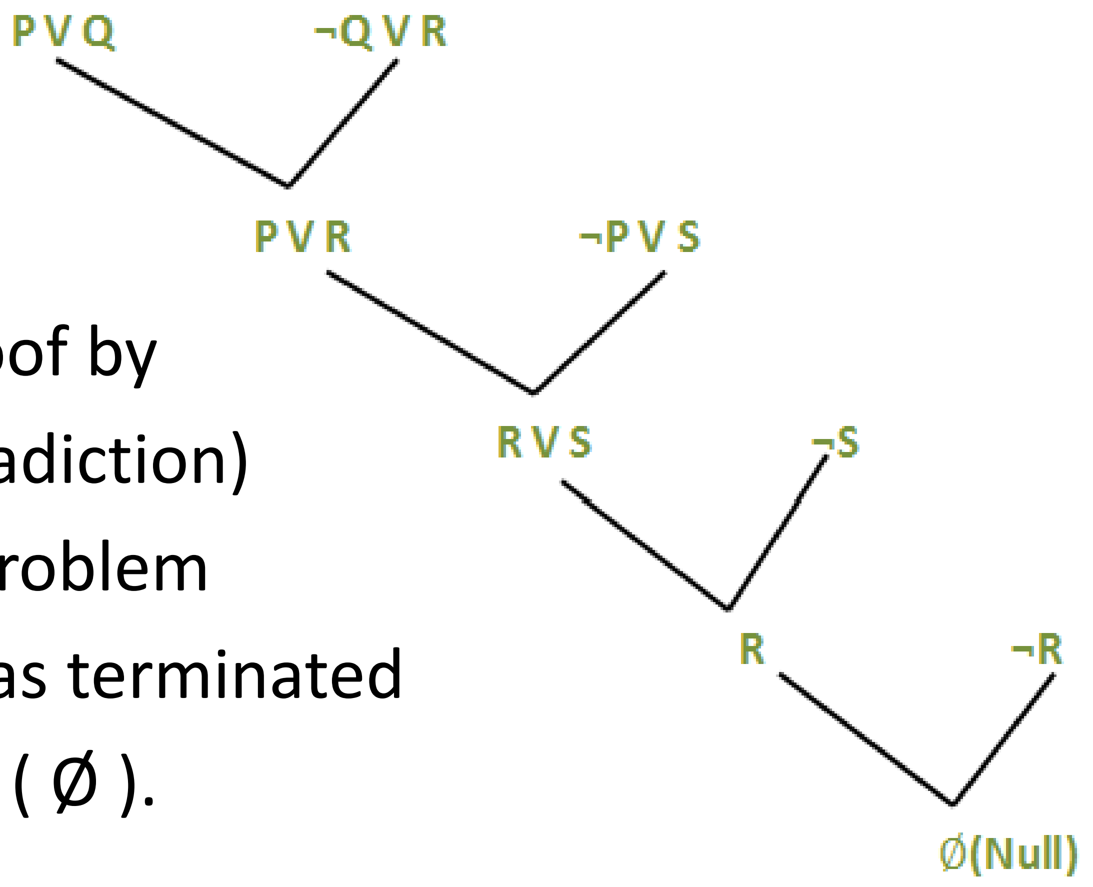
In (2), Q $\rightarrow$ R will be converted as (¬Q V R)

In (3), P $\rightarrow$ S will be converted as (¬P V S)

Negation of Goal (¬R): It will not rain.

Finally, apply the rule as shown in the following:

Resolution: Example 2



After applying Proof by Refutation (Contradiction) on the goal, the problem is solved, and it has terminated with a Null clause (Ø).

Hence, the goal is achieved. Thus, It is not raining.

Resolution: Example 3

Consider the following knowledge base:

1. Gita likes all kinds of food.
2. Mango and chapati are food.
3. Gita eats almond and is still alive.
4. Anything eaten by anyone and is still alive is food.

Goal: Gita likes almond.

Apply resolution method to prove that the goal is derivable from the given knowledge base.

Resolution: Example 3

Solution:

Convert the given sentences into FOPL as:

Let, x be the light sleeper.

1. ∀x: food(x) → likes(Gita,x)
2. food(Mango),food(chapati)
3. ∀x∀y: eats(x,y) Λ ¬ killed(x → food(y)
4. eats(Gita, almonds) Λ alive(Gita)
5. ∀x: ¬killed(x) → alive(x)
6. ∀x: alive(x) → ¬killed(x)

Resolution: Example 3

Goal: likes(Gita, almond)

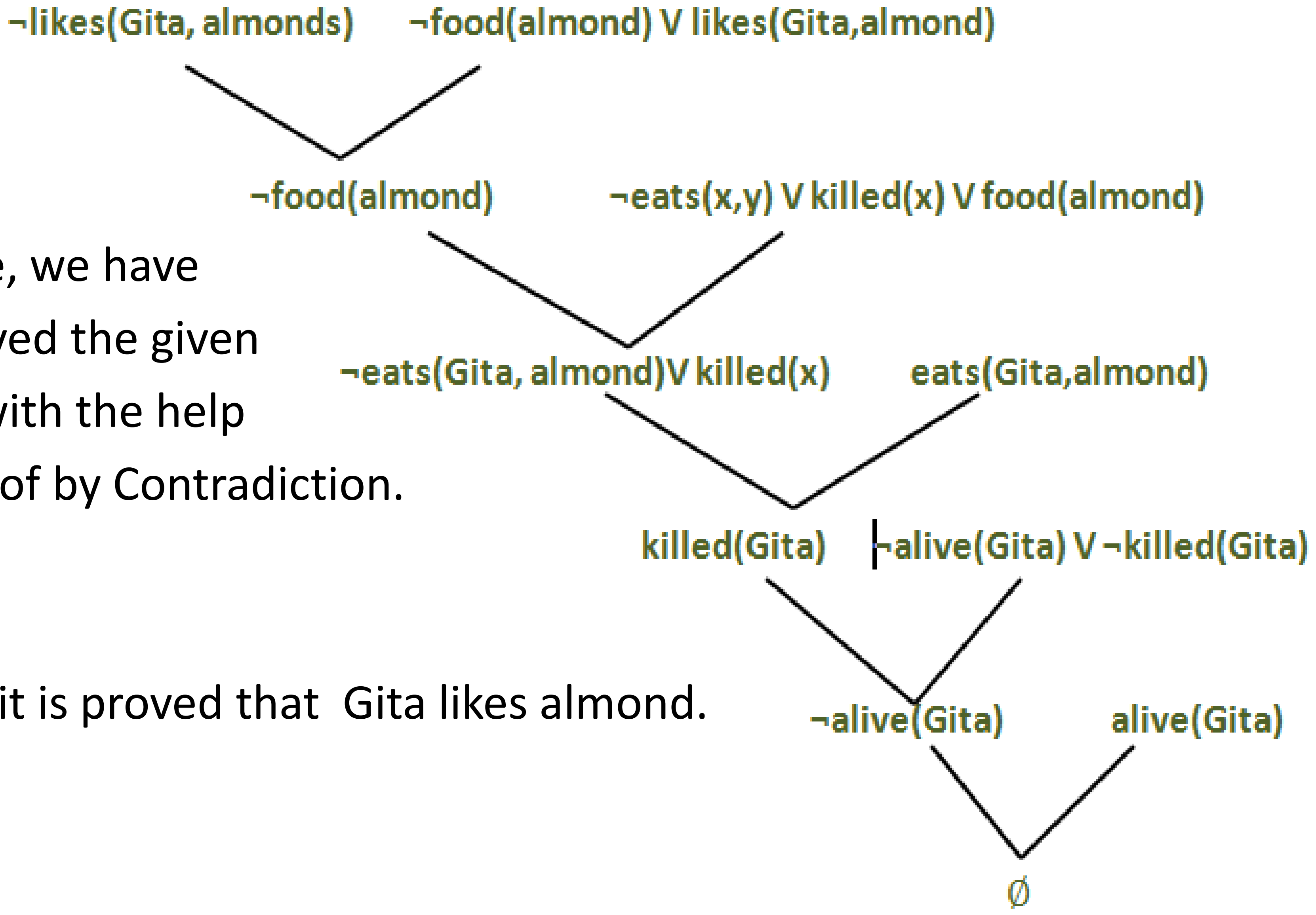
Negated goal: ¬likes(Gita, almond)

Now, rewrite in CNF form:

1. ¬food(x) V likes(Gita, x)
2. food(Mango),food(chapati)
3. ¬eats(x,y) V killed(x) V food(y)
4. eats(Gita, almonds), alive(Gita)
5. killed(x) V alive(x)
6. ¬alive(x) V ¬killed(x)

Resolution: Example 3

Finally, construct the resolution graph:


¬likes(Gita, almonds)
¬food(almond) V likes(Gita,almond)
¬food(almond)
¬eats(x,y) V killed(x) V food(almond)
¬eats(Gita, almond)V killed(x)
eats(Gita,almond)
killed(Gita)
¬alive(Gita) V ¬killed(Gita)
¬alive(Gita)
alive(Gita)
∅

Hence, we have achieved the given goal with the help of Proof by Contradiction.

Thus, it is proved that Gita likes almond.

Propositional Knowledge

Propositional Knowledge is information or understanding that can be represented in natural language or a more formal language such as mathematics and propositional logic.

It is often contrasted with knowledge that is difficult to encode in a language.

Examples: France is a country.

$x + y > 20$

The sun is larger than the earth.

Rule Based Systems

Instead of representing knowledge in a relatively declarative, static way (as a bunch of things that are true), rule-based system represent knowledge in terms of a bunch of rules that tell you what you should do or what you could conclude in different situations.

A rule-based system consists of a bunch of IF-THEN rules, a bunch of facts, and some interpreter controlling the application of the rules, given the facts.

Hence, theses are also sometimes referred to as production systems.

Rule Based Systems

There are two broad kinds of rule system: forward chaining systems, and backward chaining systems.

In a forward chaining system you start with the initial facts, and keep using the rules to draw new conclusions (or take certain actions) given those facts.

In a backward chaining system you start with some hypothesis (or goal) you are trying to prove, and keep looking for rules that would allow you to conclude that hypothesis, perhaps setting new sub goals to prove as you go.

Forward reasoning: Conflict resolution

A number of conflict resolution strategies are typically used to decide which rule to fire. These include:

- Don't fire a rule twice on the same data.
- Fire rules on more recent working memory elements before older ones. This allows the system to follow through a single chain of reasoning, rather than keeping on drawing new conclusions from old data.

Forward reasoning: Conflict resolution

- Fire rules with more specific preconditions before ones with more general preconditions. This allows us to deal with non-standard cases.

If for example:

we have a rule ``IF (bird X) THEN ADD (flies X)'' and another rule ``IF (bird X) AND (penguin X) THEN ADD (swims X)'' and a penguin called tweety

then we would fire the second rule first and start to draw conclusions from the fact that tweety swims.

Forward reasoning: properties

- It is a down-up approach, as it moves from bottom to top.
- It is a process of making a conclusion based on known facts or data, by starting from the initial state and reaches the goal state.
- This approach is also called as data-driven as we reach to the goal using available data.
- Forward reasoning approach is commonly used in the expert system, such as CLIPS, business, and production rule systems.

Backward reasoning: Use of backtracking

In backward reasoning, a proof of a theorem is constructed by trying to obtain a set of axioms from the theorem by backward application of inference rules.

In other words, this approach starts with the desired conclusion and works backward to find supporting facts. Therefore, it is also known as Goal-Driven Approach.

Backward reasoning can be automated if, for each instance of a rule, the premises are uniquely determined by the conclusion.

Backward reasoning: properties

- It is known as a top-down approach.
- Backward reasoning is based on modus ponens inference rule.
- In backward reasoning, the goal is broken into sub-goal or sub-goals to prove the facts true.
- It is called a goal-driven approach, as a list of goals decides which rules are selected and used.

Backward reasoning: properties

- Backward reasoning algorithm is used in game theory, automated theorem proving tools, inference engines, proof assistants, and various AI applications.

- The backward reasoning method mostly used a depth-first search strategy for proof.

Structured KR: Semantic Nets

A semantic network is often used as a form of knowledge representation.

It is a directed graph consisting of vertices which represent concepts and edges which represent semantic relations between the concepts.

The following semantic relations are commonly represented in a semantic net:

Meronymy (A is part of B)
Holonymy (B has A as a part of itself)

Semantic Nets

The following semantic relations are commonly represented in a semantic net (continued):

Hyponymy (or troponymy) (A is subordinate of B; A is kind of B)

Hypernymy (A is superordinate of B)

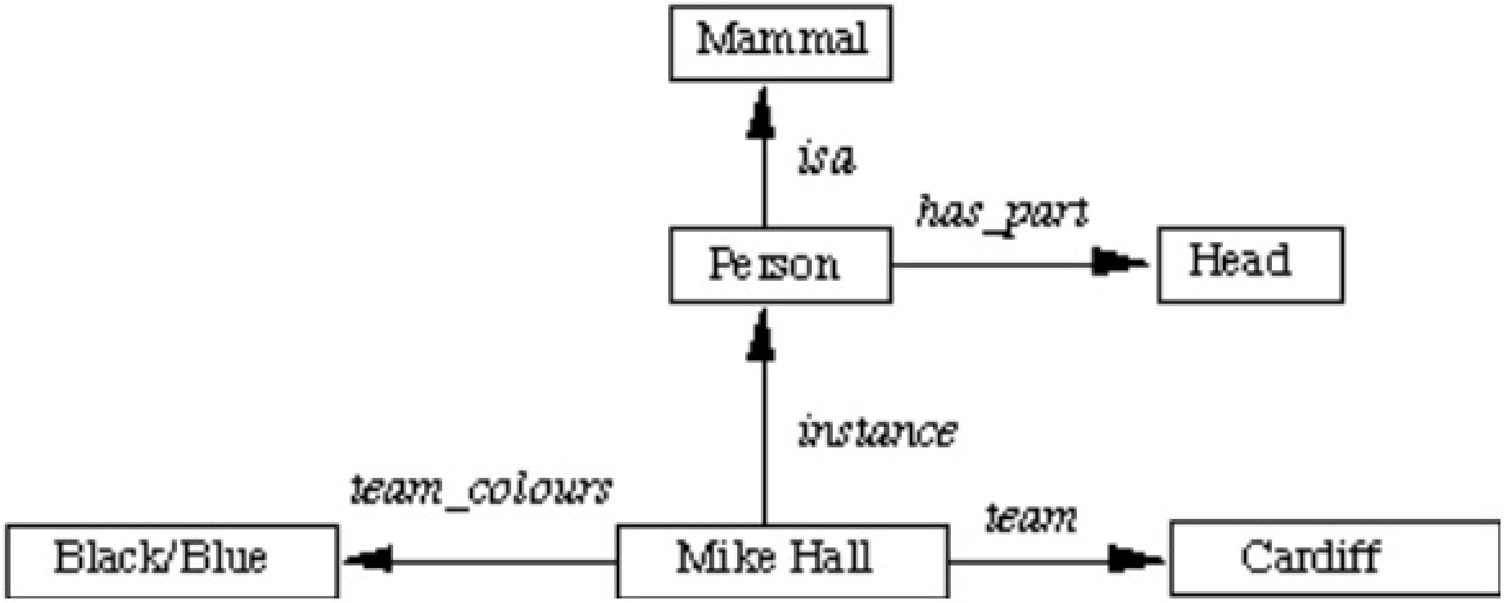
Synonymy (A denotes the same as B)

Antonymy (A denotes the opposite of B)

Semantic Nets: Example

Example:

The physical attributes of a person can be represented as in the following figure using a semantic net.



Semantic Nets: Example

These values can also be represented in logic as: isa(person, mammal), instance(MikeHall, person) team(Mike-Hall, Cardiff)

conventional predicates such as lecturer(dave) can be written as instance (dave, lecturer)

isa and *instance* represent inheritance and are popular in many knowledge representation schemes.

Semantic Nets: Example

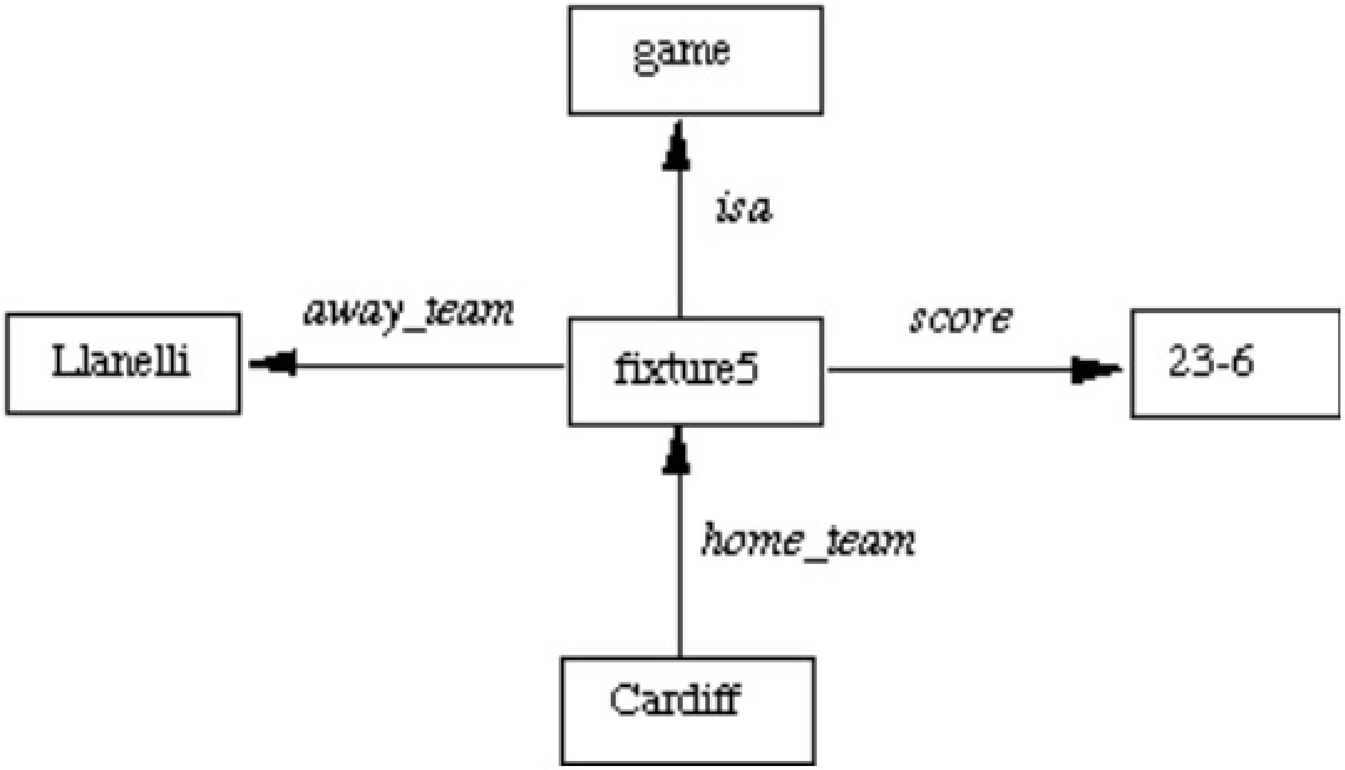
But we have a problem: How we can have more than 2 place predicates in semantic nets?

E.g. score(Cardiff, Llanelli, 23-6)

Solution:

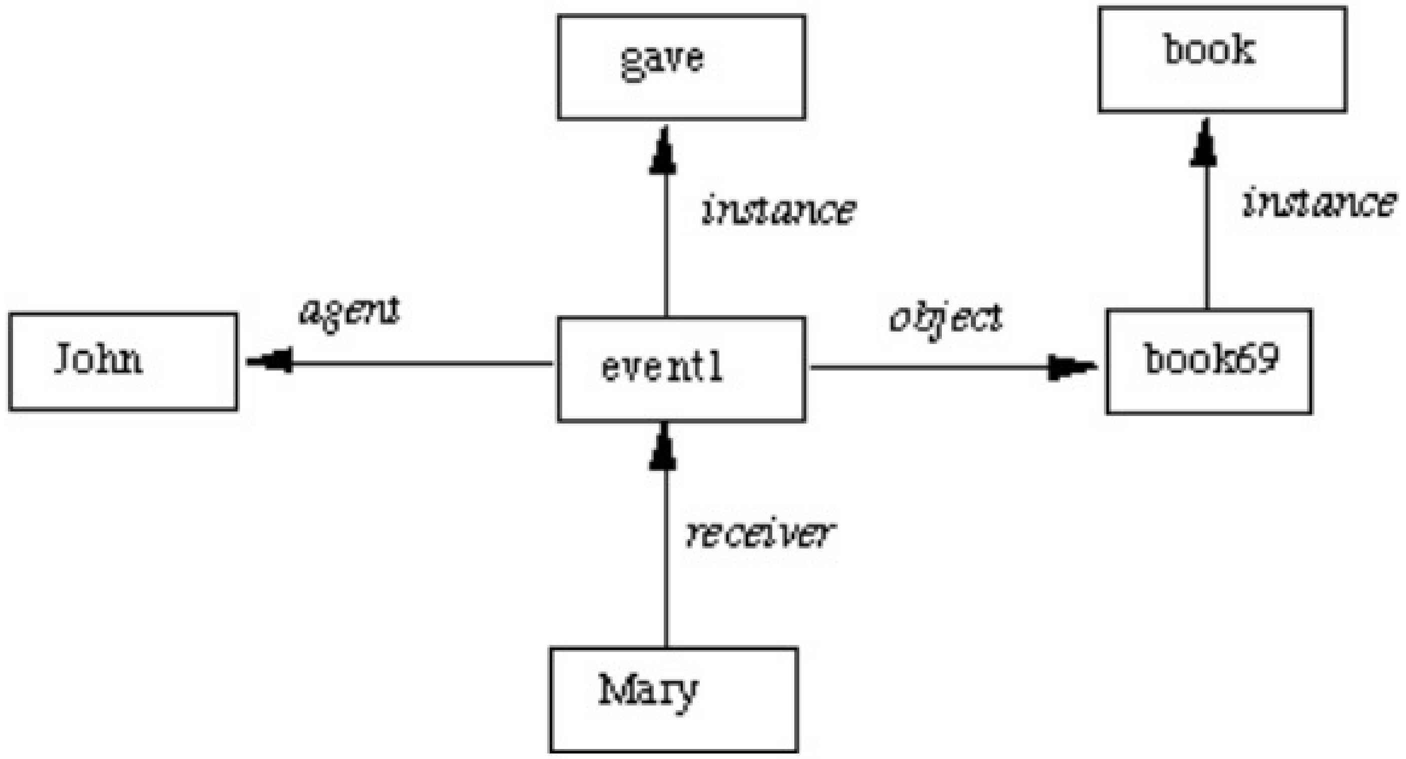
- Create new nodes to represent new objects either contained or alluded to in the knowledge, game and fixture in the current example.
- Relate information to nodes and fill up slots.

Semantic Nets: Example



As a more complex example consider the sentence: John gave Mary the book. Here we have several aspects of an event.

Semantic Nets: Example



Semantic Net - Slots

A slot is a relation that maps from its domain of classes to its range of values.

A relation is a set of ordered pairs so one relation is a subset of another.

Since slot is a set the set of all slots can be represent by a metaclass called Slot.

Semantic Net – Slots Example

Example 1: Coach

instance:	SLOT
domain:	Rugby-Team
range:	Person
range-constraint:	(experience x.manager)
default:	
single-valued:	TRUE

Example 2: *Colour*

instance:	SLOT
domain:	Physical-Object
range:	Colour-Set
single-valued:	FALSE

Semantic Net – Slots Example

Example3 : Team-Colours

instance:	SLOT
isa:	Colour
domain:	team-player
range:	Colour-Set
range-constraint:	not Pink
single-valued:	FALSE

Example 4: Position

instance:	SLOT
domain:	Rugby-Player
range:	{ Back, Forward, Reserve }
to-compute:	x.position
single-valued:	TRUE

Semantic Net - Inheritance

Inheritance is one of the main kind of reasoning done in semantic nets.

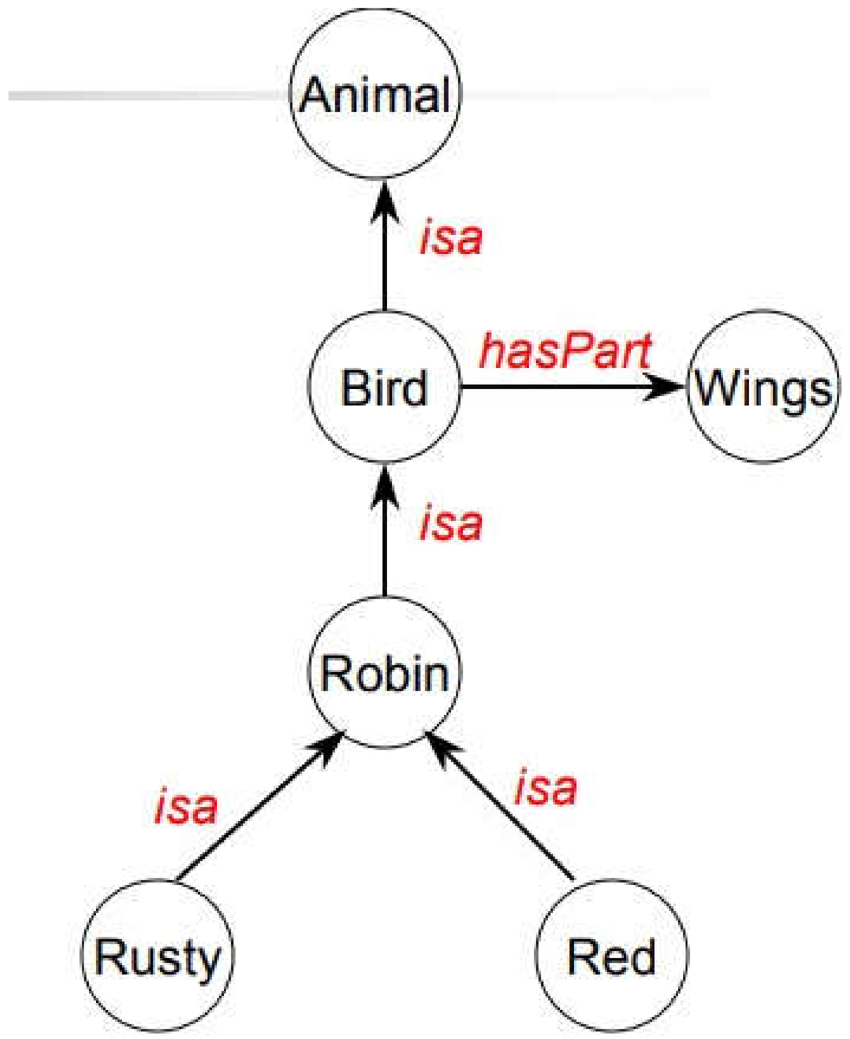
The ISA (is a) relation is often used to link a class and its super- class.

Some links (e.g. haspart) are inherited along ISA paths.

The semantics of a semantic net can be relatively informal or very formal (Often defined at the implementation level).

Semantic Net – Inheritance

Example:



Semantic Net - Frames

Frames are descriptions of conceptual individuals.

Frames can exist for ``real'' objects such as ``The Watergate Hotel'', sets of objects such as ``Hotels'', or more ``abstract'' objects such as ``Cola-Wars'' or ``Watergate''.

Frames are essentially defined by their relationships with other frames.

Relationships between frames are represented using slots. If a frame *f* is in a relationship *r* to a frame *g*, then we put the value *g* in the *r* slot of *f.*

Semantic Net - Frames

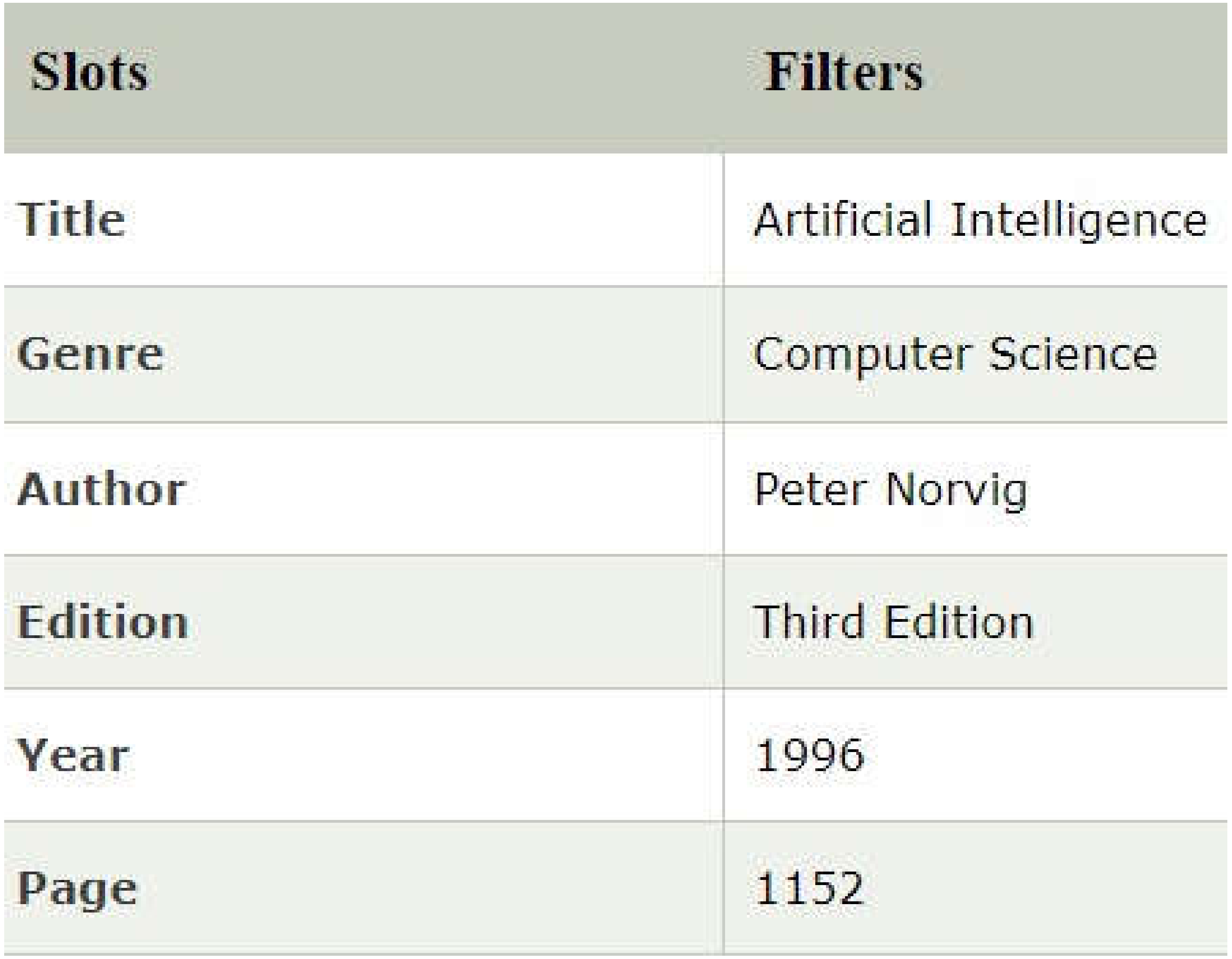
Facets are features of frames which enable us to put constraints on the frames. Example: IF-NEEDED facts are called when data of any particular slot is needed.

A frame may consist of any number of slots, and a slot may include any number of facets and facets may have any number of values.

A frame is also known as slot-filter knowledge representation in artificial intelligence.

Semantic Net - Frames

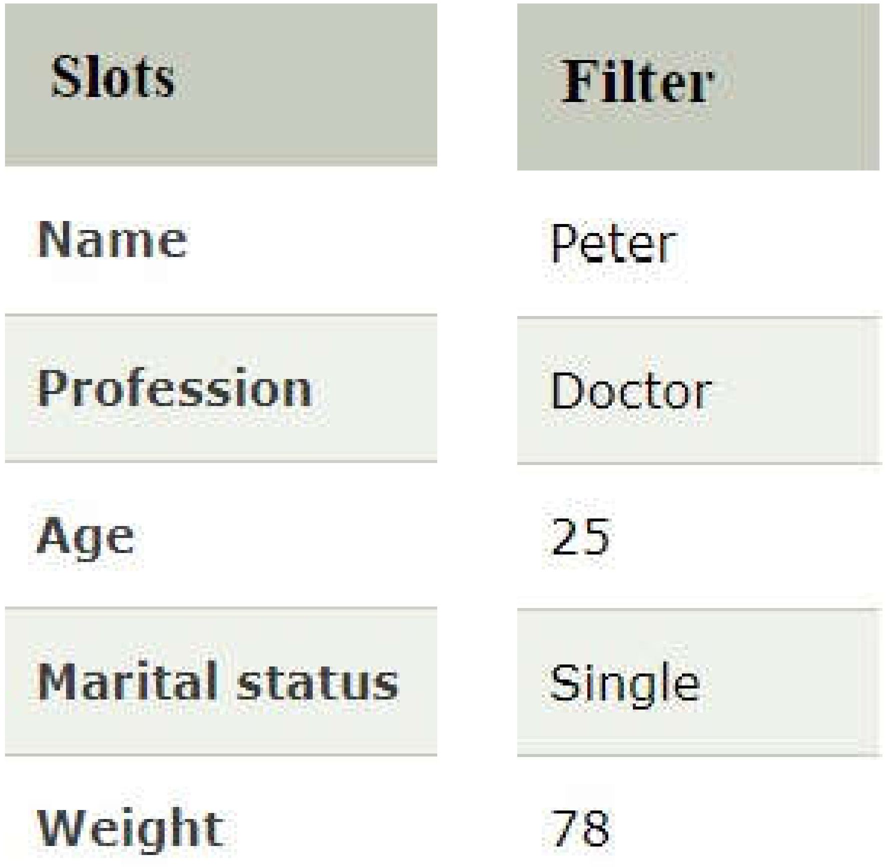
Let's take an example of a frame for a book:



Slots	Filters
Title	Artificial Intelligence
Genre	Computer Science
Author	Peter Norvig
Edition	Third Edition
Year	1996
Page	1152

Semantic Net - Frames

Let's suppose we are taking an entity, Peter. Peter is an engineer as a profession, and his age is 25, he lives in city London, and the country is England. So following is the frame representation for this:



Slots	Filter
Name	Peter
Profession	Doctor
Age	25
Marital status	Single
Weight	78

Semantic Net - Frames

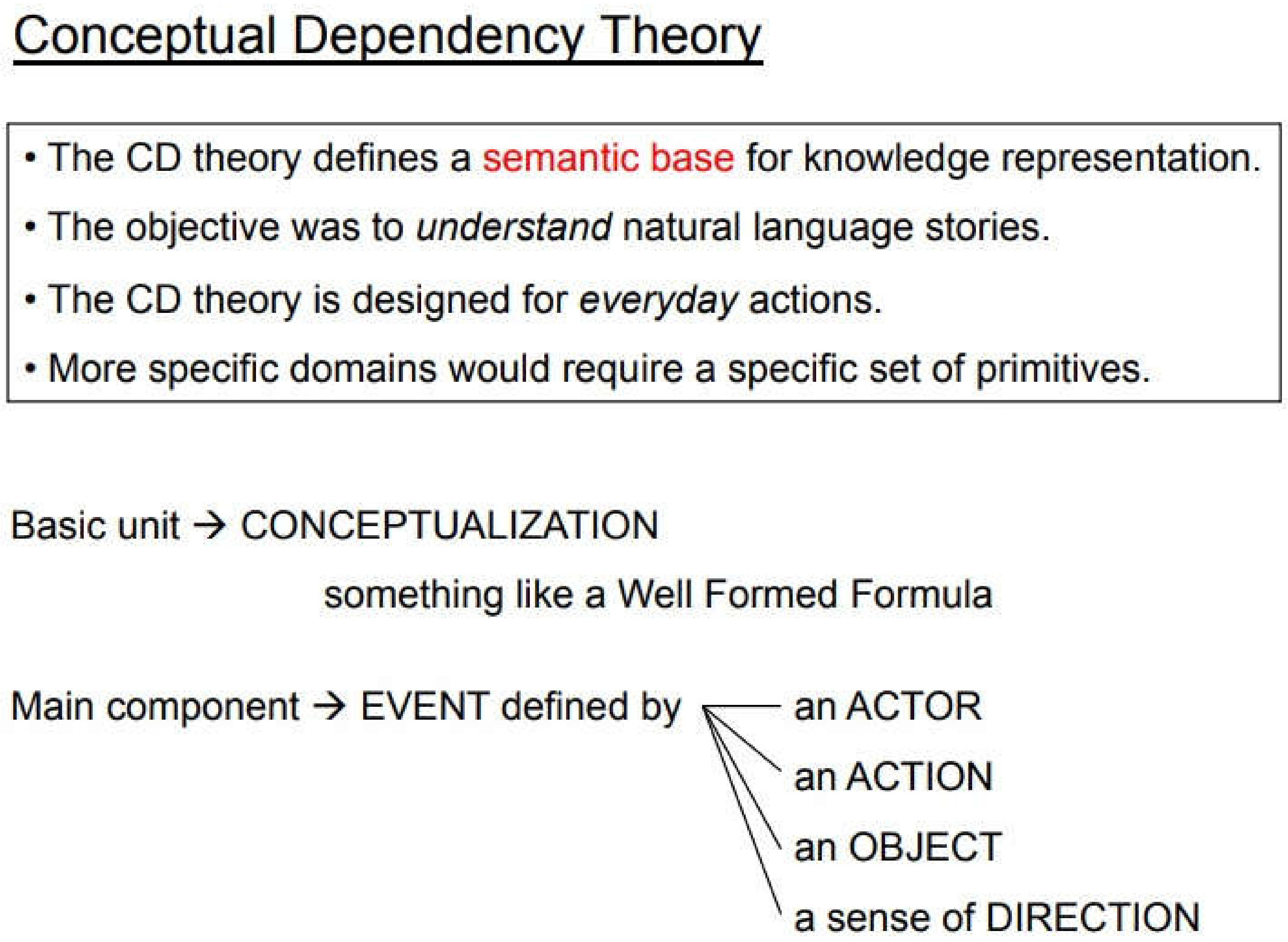
A single frame is not much useful.

Frames system consist of a collection of frames which are connected.

In the frame, knowledge about an object or event can be stored together in the knowledge base.

The frame is a type of technology which is widely used in various applications including Natural language processing and machine visions.

Conceptual Dependency formalism



Conceptual Dependency Theory

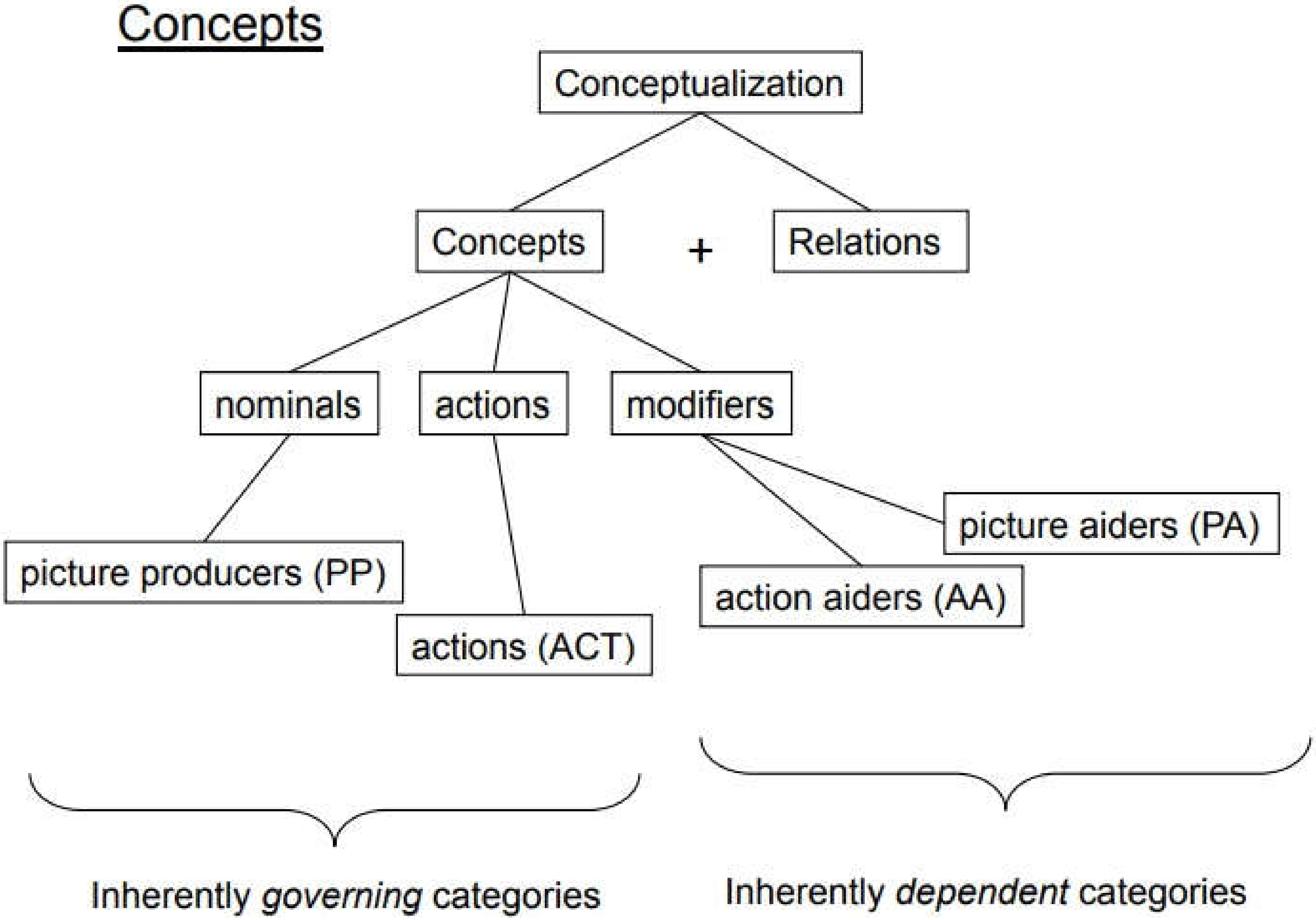
- The CD theory defines a semantic base for knowledge representation.
- The objective was to *understand* natural language stories.
- The CD theory is designed for *everyday* actions.
- More specific domains would require a specific set of primitives.

Basic unit → CONCEPTUALIZATION

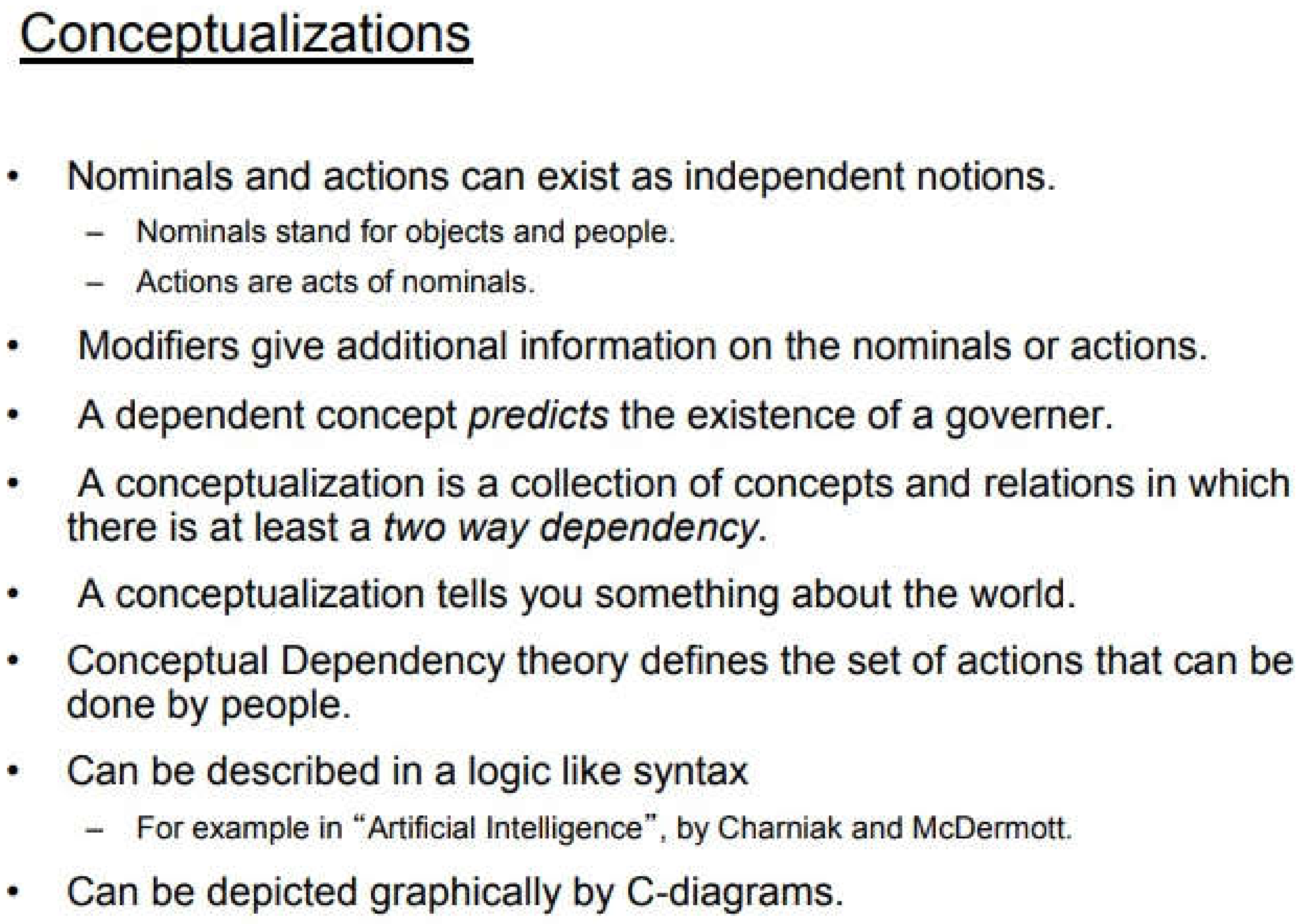
something like a Well Formed Formula

Main component → EVENT defined by
- an ACTOR
- an ACTION
- an OBJECT
- a sense of DIRECTION

Conceptual Dependency formalism



Conceptual Dependency formalism

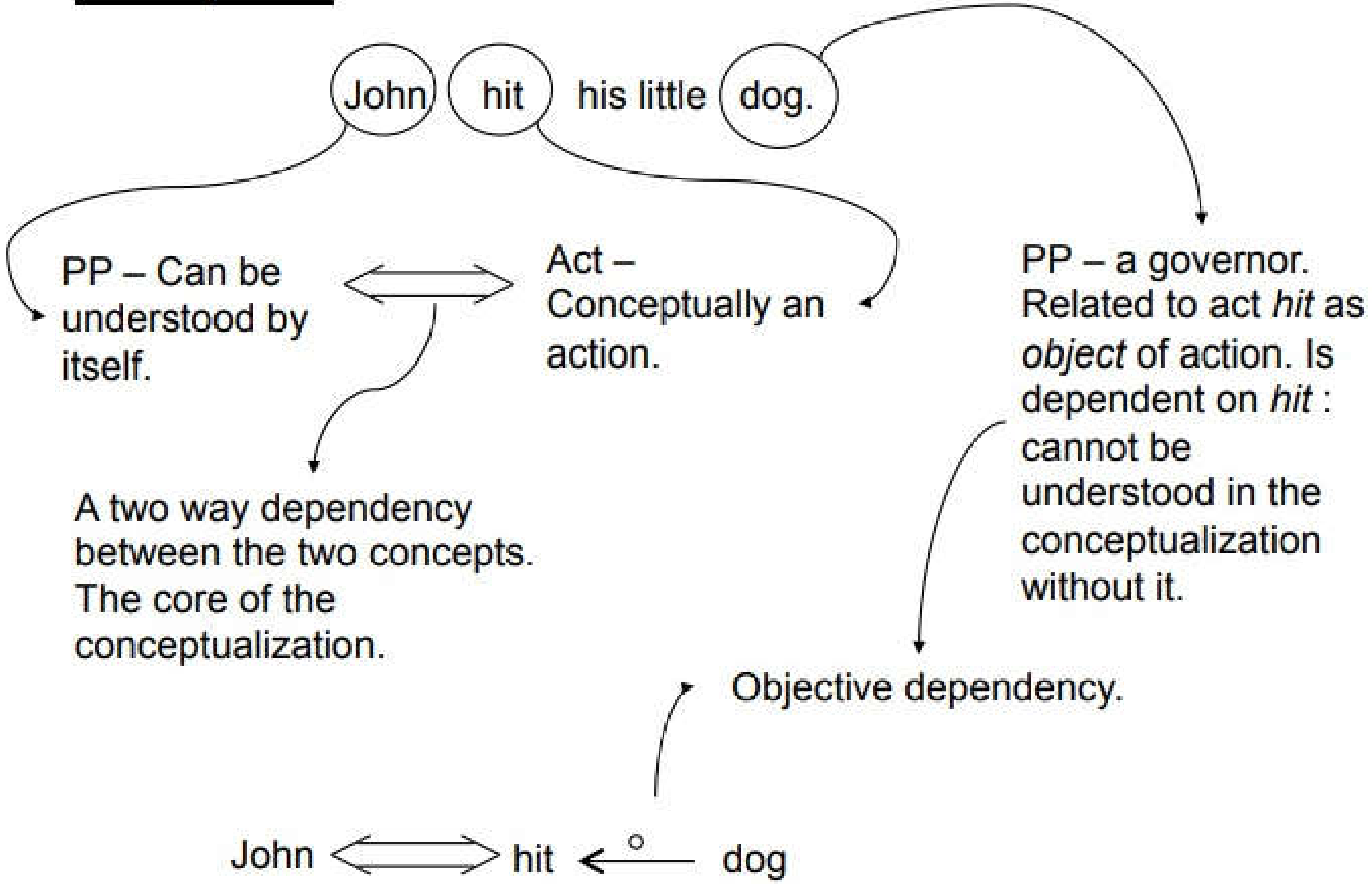


Conceptualizations

- Nominals and actions can exist as independent notions.
  - Nominals stand for objects and people.
  - Actions are acts of nominals.
- Modifiers give additional information on the nominals or actions.
- A dependent concept *predicts* the existence of a governer.
- A conceptualization is a collection of concepts and relations in which there is at least a *two way dependency*.
- A conceptualization tells you something about the world.
- Conceptual Dependency theory defines the set of actions that can be done by people.
- Can be described in a logic like syntax
  - For example in "Artificial Intelligence", by Charniak and McDermott.
- Can be depicted graphically by C-diagrams.

Conceptual Dependency formalism

Example 1.



PP – Can be understood by itself.

Act – Conceptually an action.

PP – a governor. Related to act *hit* as *object* of action. Is dependent on *hit* : cannot be understood in the conceptualization without it.

A two way dependency between the two concepts. The core of the conceptualization.

Objective dependency.

$$\text{John} \Longleftrightarrow \text{hit} \xleftarrow{o} \text{dog}$$

Conceptual Dependency formalism

Example 1 (continued)

John hit his little dog.

A PA dependent on the dog. *Attributive Dependency.*

$\longleftarrow$

A little more complex - "his" is dependent on "dog". But it is also a linguistic item – pronoun – for "John". One PP as *dependent* on another PP : *Prepositional Dependency.* Label indicates type of dependency.

$\xLeftarrow{\text{Poss-by}}$

$$\begin{tikzcd}
\text{John} \arrow[r, Leftrightarrow] & \text{hit} & & \text{dog} \arrow[ll, "o"'] & \\
& & \text{little} \arrow[ur] & & \text{John} \arrow[ul, Rightarrow, "\text{Poss-by}"']
\end{tikzcd}$$

Conceptual Dependency formalism

Conceptual Dependencies

1. $\text{PP} \Longleftrightarrow \text{ACT}$

Certain PPs can ACT

2. $\text{PP} \Lleftarrow\!\!\Rrightarrow \text{PA}$

PPs (and some Conceptualizations) can be described by an attrtibute.

3. $\text{ACT} \xleftarrow{o} \text{PP}$

ACTs have objects.

4. $$\begin{tikzcd}[row sep=small] & & \text{LOC} \\ \text{ACT} & \cdot \arrow[l, "D"'] \arrow[ur] & \\ & & \text{LOC} \arrow[ul] \end{tikzcd}$$

ACTs have direction.

5. $$\begin{tikzcd}[row sep=small] & & \text{PP} \\ \text{ACT} & \cdot \arrow[l, "R"'] \arrow[ur] & \\ & & \text{PP} \arrow[ul] \end{tikzcd}$$

ACTs have recipients..

6. $\text{ACT} \xleftarrow{o} \Updownarrow$

MTRANS requires conceptualizations as objects, and MBUILD has its own object type.

Conceptual Dependency formalism

Conceptual Dependencies

7. $$\begin{tikzcd} \text{ACT} & \Updownarrow \arrow[l, "I"'] \end{tikzcd}$$

ACTs have conceptualizations as instruments.

8. $$\begin{tikzcd} \text{PP} \arrow[d, leftrightarrow] & \text{PP} \arrow[d, leftrightarrow] \\ \Leftrightarrow & \Leftrightarrow \end{tikzcd}$$

PPs can be described by the conceptualizations in which they occur.

9. $$\begin{tikzcd} \text{T} \arrow[d] \\ \Leftrightarrow \end{tikzcd}$$

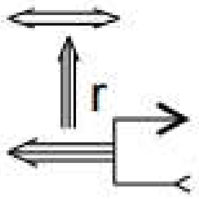
Conceptualizations have times.

10. $$\begin{tikzcd} \text{LOC} \arrow[d, Rightarrow] \\ \Leftrightarrow \end{tikzcd}$$

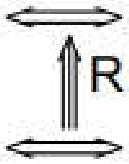
Conceptualizations have locations.

Conceptual Dependency formalism

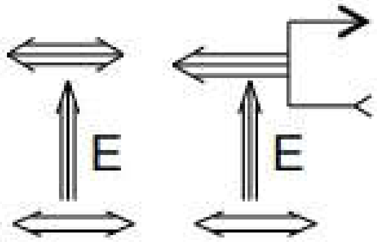
Conceptual Dependencies

11. 

Conceptualizations can *result* in state changes for PPs.

12. 

Conceptualizations involving mental ACTs can server as *reasons* for conceptualizations.

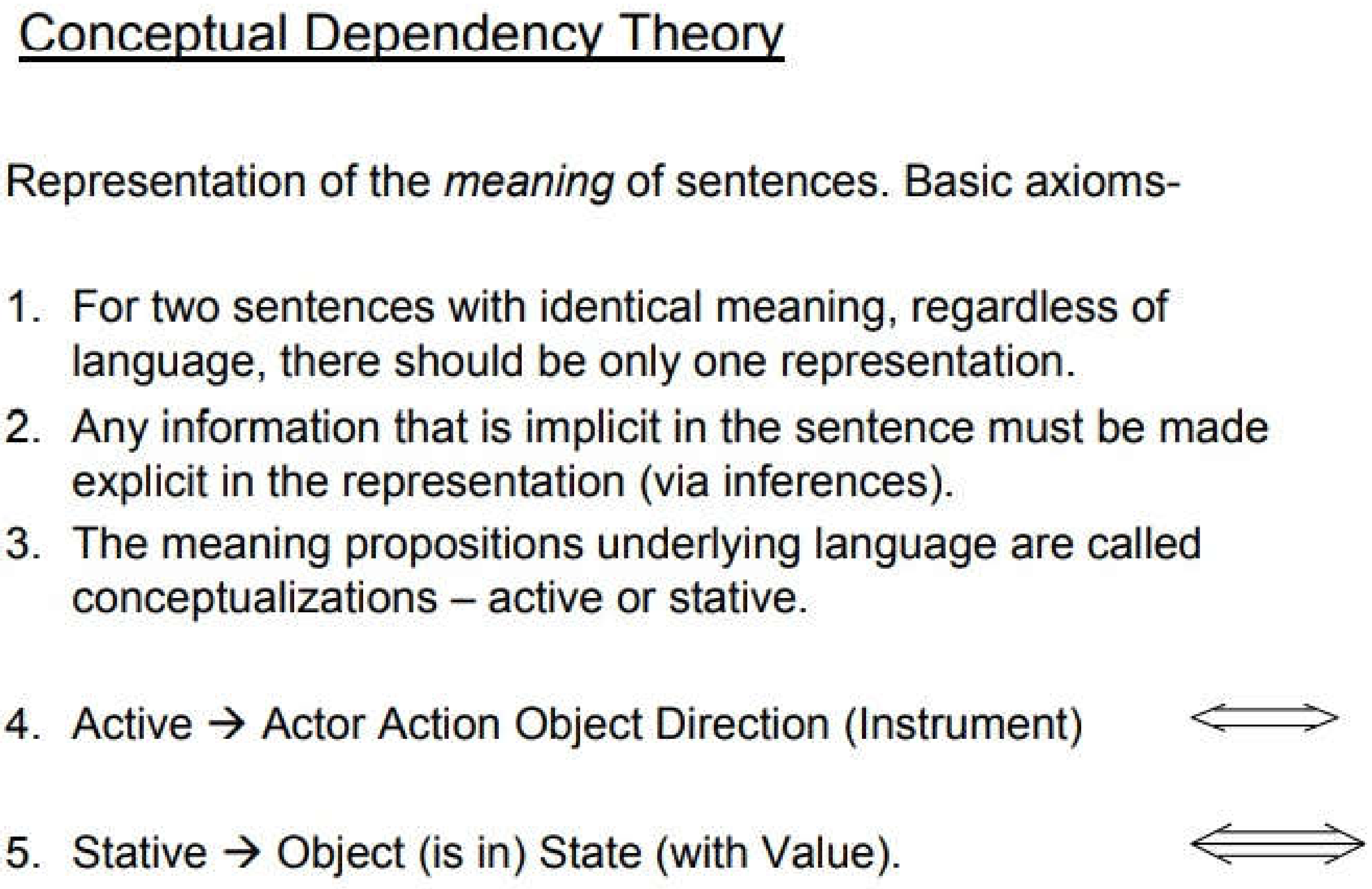
13. 

State or state changes can *enable* conceptualizations to occur.

14. PP $\Longleftrightarrow$ PP
One PP is equivalent to or an instance of another PP.

15. $$\begin{tikzcd} \text{ACT} \\ \text{AA} \arrow[u] \end{tikzcd}$$
ACTs can be varied along certain dimensions (e.g. speed for motions ACTS).

Conceptual Dependency formalism



Conceptual Dependency Theory

Representation of the *meaning* of sentences. Basic axioms-

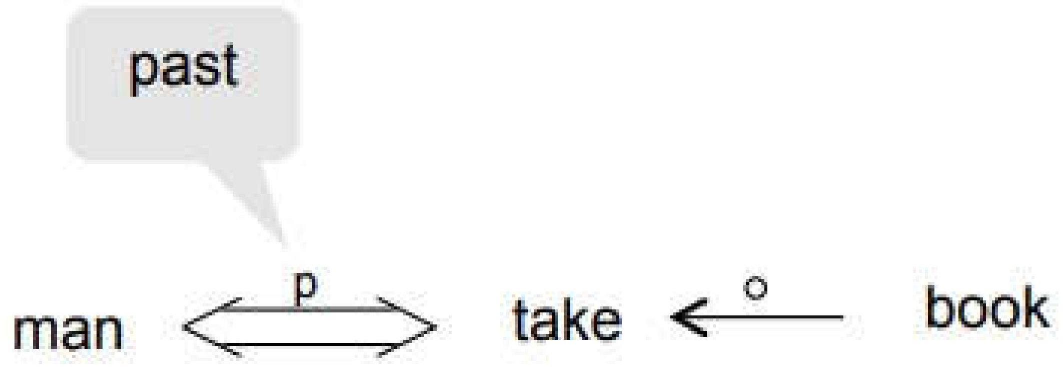
1. For two sentences with identical meaning, regardless of language, there should be only one representation.
2. Any information that is implicit in the sentence must be made explicit in the representation (via inferences).
3. The meaning propositions underlying language are called conceptualizations – active or stative.

4. Active → Actor Action Object Direction (Instrument) $\Leftrightarrow$

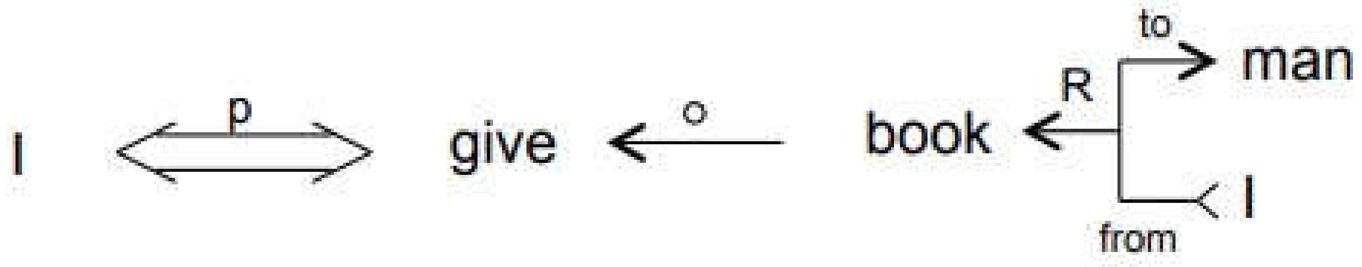
5. Stative → Object (is in) State (with Value). $\Lleftarrow\!\!\Rrightarrow$

Conceptual Dependency formalism

Example: The man took a book.



Example: I gave the man a book.



Other knowledge representations

Inheritable Knowledge

Inferential Knowledge

Procedural knowledge

Inheritable Knowledge

Relational knowledge is made up of objects consisting of

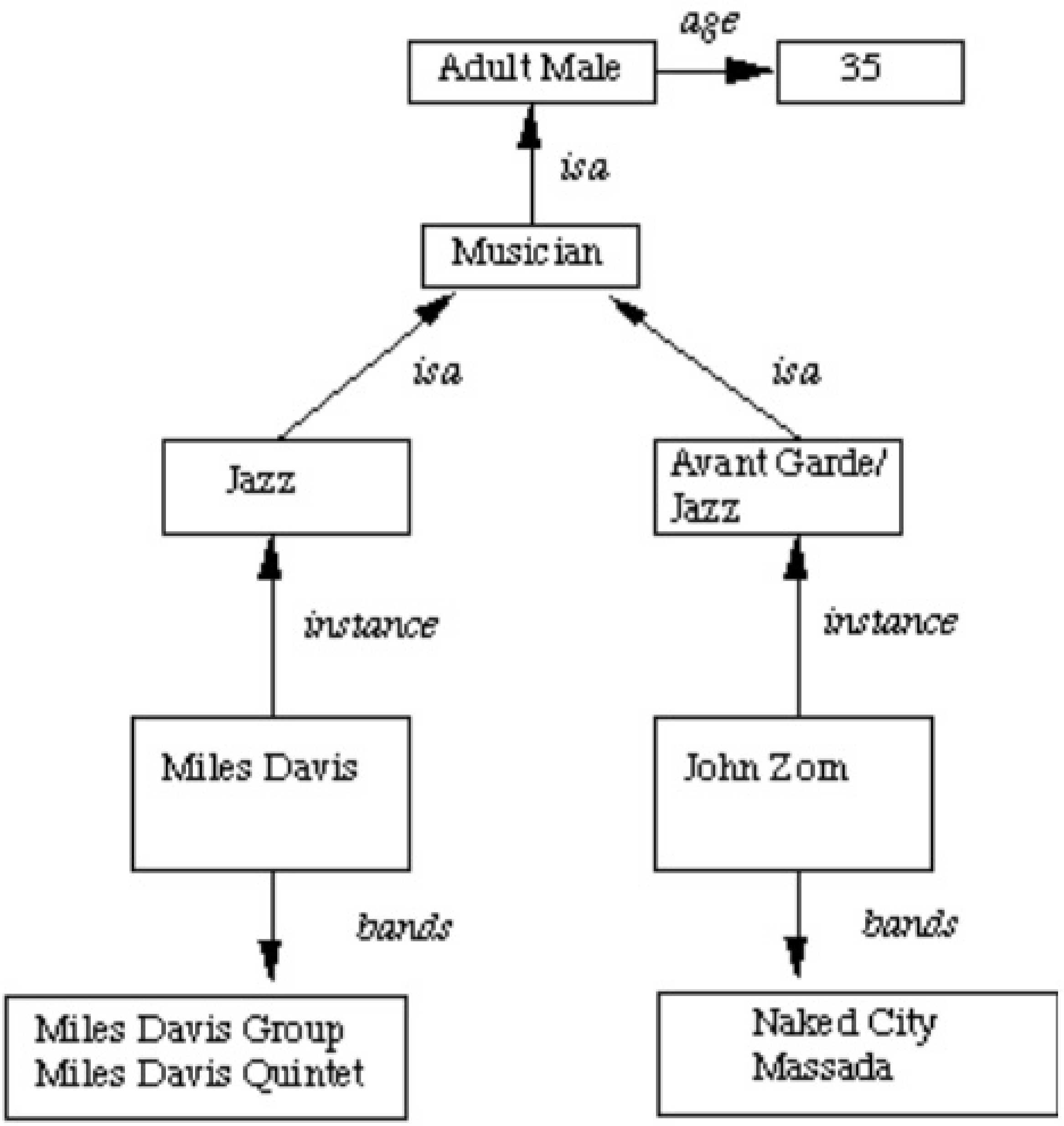
- Attributes
- Corresponding associated values

We extend the base more by allowing inference mechanisms:

Property inheritance:

- Elements inherit values from being members of a class.
- Data must be organised into a hierarchy of classes.

Inheritable Knowledge



Inheritable Knowledge

Boxed nodes -- objects and values of attributes of objects.

Values can be objects with attributes and so on.

Arrows -- point from object to its value.

This structure is known as a slot and filler structure, semantic network or a collection of frames.

Inferential Knowledge

Represent knowledge as formal logic:

Advantages:

- A set of strict rules.
  - o Can be used to derive more facts.
  - o Truths of new statements can be verified.
  - o Guaranteed correctness.
- Many inference procedures available to in implement standard rules of logic.
- Popular in AI systems. e.g Automated theorem proving.

Procedural knowledge

Knowledge encoded in some procedures.

Small programs that know how to do specific things, how to proceed.

Ex: A parser in a natural language understander has the knowledge that a noun phrase may contain articles, adjectives and nouns.

It is represented by calls to routines that know how to process articles, adjectives and nouns.

Procedural knowledge

Advantages:

- Heuristic or domain specific knowledge can be represented.

- Extended logical inferences, such as default reasoning facilitated.

- Side effects of actions may be modelled. Some rules may become false in time. Keeping track of this in large systems may be tricky.