

Knowledge Representation

Unification

-
- Unification is a process of making two different logical atomic expressions identical by finding a substitution. Unification depends on the substitution process.
 - It takes two literals as input and makes them identical using substitution.
 - Let Ψ_1 and Ψ_2 be two atomic sentences and σ be a unifier such that, $\Psi_1\sigma = \Psi_2\sigma$, then it can be expressed as **UNIFY**(Ψ_1, Ψ_2).
 - Example: Find the MGU for Unify{King(x), King(John)}
-

-
- Let $\Psi_1 = \text{King}(x)$, $\Psi_2 = \text{King}(\text{John})$,

Substitution $\theta = \{\text{John}/x\}$ is a unifier for these atoms and applying this substitution, and both expressions will be identical.

- The UNIFY algorithm is used for unification, which takes two atomic sentences and returns a unifier for those sentences (If any exist).
- Unification is a key component of all first-order inference algorithms.
- It returns fail if the expressions do not match with each other.
- The substitution variables are called Most General Unifier or MGU.

E.g. Let's say there are two different expressions, **$P(x, y)$** , and **$P(a, f(z))$** .

In this example, we need to make both above statements identical to each other. For this, we will perform the substitution.

$P(x, y)$ (i)

$P(a, f(z))$ (ii)

- Substitute x with a , and y with $f(z)$ in the first expression, and it will be represented as **a/x** and **$f(z)/y$** .
 - With both the substitutions, the first expression will be identical to the second expression and the substitution set will be: **$[a/x, f(z)/y]$** .
-

Conditions for Unification:

- Predicate symbol must be same, atoms or expression with different predicate symbol can never be unified.
 - Number of Arguments in both expressions must be identical.
 - Unification will fail if there are two similar variables present in the same expression.
-

Algorithm : Unify(Ψ_1, Ψ_2)

Step. 1: If Ψ_1 or Ψ_2 is a variable or constant, then:

- a) If Ψ_1 or Ψ_2 are identical, then return NIL.
- b) Else if Ψ_1 is a variable,
 - a. then if Ψ_1 occurs in Ψ_2 , then return FAILURE
 - b. Else return { (Ψ_2 / Ψ_1) }.
- c) Else if Ψ_2 is a variable,
 - a. If Ψ_2 occurs in Ψ_1 then return FAILURE,
 - b. Else return { (Ψ_1 / Ψ_2) }.
- d) Else return FAILURE.

Step.2: If the initial Predicate symbol in Ψ_1 and Ψ_2 are not same, then return FAILURE.

Step. 3: IF Ψ_1 and Ψ_2 have a different number of arguments, then return FAILURE.

Step. 4: Set Substitution set(SUBST) to NIL.

Step. 5: For $i=1$ to the number of elements in Ψ_1 .

- a) Call Unify function with the i th element of Ψ_1 and i th element of Ψ_2 , and put the result into S.
- b) If S = failure then returns Failure
- c) If $S \neq \text{NIL}$ then do,
 - a. Apply S to the remainder of both L1 and L2.
 - b. SUBST= APPEND(S, SUBST).

Step.6: Return SUBST.

Implementation of the Algorithm

Step.1: Initialize the substitution set to be empty.

Step.2: Recursively unify atomic sentences:

- a. Check for Identical expression match.
- b. If one expression is a variable v_i , and the other is a term t_i which does not contain variable v_i , then:
 - a. Substitute t_i / v_i in the existing substitutions
 - b. Add t_i / v_i to the substitution setlist.
- c. If both the expressions are functions, then function name must be similar, and the number of arguments must be the same in both the expression.

For each pair of the following atomic sentences find the most general unifier (If exist).

1. Find the MGU of $\{p(f(a), g(Y))$ and $p(X, X)\}$

Sol:

$S_0 \Rightarrow$ Here, $\Psi_1 = p(f(a), g(Y))$, and $\Psi_2 = p(X, X)$

SUBST $\theta = \{f(a) / X\}$

$S_1 \Rightarrow \Psi_1 = p(f(a), g(Y))$, and $\Psi_2 = p(f(a), f(a))$

SUBST $\theta = \{f(a) / g(y)\}$,

Unification failed.

2. Find the MGU of $\{p(b, X, f(g(Z)))$ and $p(Z, f(Y), f(Y))\}$

Here, $\Psi_1 = p(b, X, f(g(Z)))$, and $\Psi_2 = p(Z, f(Y), f(Y))$

$S_0 \Rightarrow \{ p(b, X, f(g(Z))); p(Z, f(Y), f(Y)) \}$

SUBST $\theta = \{b/Z\}$

$S_1 \Rightarrow \{ p(b, X, f(g(b))); p(b, f(Y), f(Y)) \}$

SUBST $\theta = \{f(Y) / X\}$

$S_2 \Rightarrow \{ p(b, f(Y), f(g(b))); p(b, f(Y), f(Y)) \}$

SUBST $\theta = \{g(b) / Y\}$

$S_2 \Rightarrow \{ p(b, f(g(b)), f(g(b))); p(b, f(g(b)), f(g(b))) \}$

Unified Successfully.

And Unifier = $\{ b/Z, f(Y) / X, g(b) / Y \}$.

3. Find the MGU of $\{p(X, X), \text{ and } p(Z, f(Z))\}$

Here, $\Psi_1 = \{p(X, X), \text{ and } \Psi_2 = p(Z, f(Z))\}$

$S_0 \Rightarrow \{p(X, X), p(Z, f(Z))\}$

SUBST $\theta = \{X/Z\}$

$S_1 \Rightarrow \{p(Z, Z), p(Z, f(Z))\}$

SUBST $\theta = \{f(Z) / Z\}$

Hence, unification is not possible for these expressions.

4. Find the MGU of UNIFY(prime (11), prime(y))

Here, $\Psi_1 = \{\text{prime}(11)\}$, and $\Psi_2 = \{\text{prime}(y)\}$

$S_0 \Rightarrow \{\text{prime}(11), \text{prime}(y)\}$

SUBST $\theta = \{11/y\}$

$S_1 \Rightarrow \{\text{prime}(11), \text{prime}(11)\}$

Successfully unified.

Unifier: $\{11/y\}$.

5. Find the MGU of $Q(a, g(x, a), f(y)), Q(a, g(f(b), a), x)$

Here, $\Psi_1 = Q(a, g(x, a), f(y))$, and $\Psi_2 = Q(a, g(f(b), a), x)$

$S_0 \Rightarrow \{Q(a, g(x, a), f(y)); Q(a, g(f(b), a), x)\}$

SUBST $\theta = \{f(b)/x\}$

$S_1 \Rightarrow \{Q(a, g(f(b), a), f(y)); Q(a, g(f(b), a), f(b))\}$

SUBST $\theta = \{b/y\}$

$S_1 \Rightarrow \{Q(a, g(f(b), a), f(b)); Q(a, g(f(b), a), f(b))\},$

Successfully Unified.

Unifier: $[a/a, f(b)/x, b/y]$.

RESOLUTION IN FOL

-
- Resolution is a theorem proving technique that proceeds by building refutation proofs, i.e., proofs by contradictions. It was invented by a Mathematician John Alan Robinson in the year 1965.
 - Resolution is used, if there are various statements are given, and we need to prove a conclusion of those statements. Unification is a key concept in proofs by resolutions. Resolution is a single inference rule which can efficiently operate on the **conjunctive normal form or clausal form**.
 - **Clause**: Disjunction of literals (an atomic sentence) is called a **clause**. It is also known as a unit clause.
 - **Conjunctive Normal Form**: A sentence represented as a conjunction of clauses is said to be **conjunctive normal form** or **CNF**.
-

The resolution inference rule

- The resolution rule for first-order logic is simply a lifted version of the propositional rule.
- Resolution can resolve two clauses if they contain complementary literals, which are assumed to be standardized apart so that they share no variables.

$$\frac{l_1 \vee \dots \vee l_k, \quad m_1 \vee \dots \vee m_n}{\text{SUBST}(\theta, l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n)}$$

- Where l_i and m_j are complementary literals.
 - This rule is also called the **binary resolution rule** because it only resolves exactly two literals.
-

Example :

We can resolve two clauses which are given below:

$[\text{Animal}(g(x) \vee \text{Loves}(f(x), x)]$ and $[\neg \text{Loves}(a, b) \vee \neg \text{Kills}(a, b)]$

Where two complimentary literals are: **Loves (f(x), x) and \neg Loves (a, b)**

These literals can be unified with unifier **$\theta = [a/f(x), \text{ and } b/x]$** , and it will generate a resolvent clause:

$[\text{Animal}(g(x) \vee \neg \text{Kills}(f(x), x)].$

Steps of Resolution :

1. Conversion of facts into first-order logic.
 2. Convert FOL statements into CNF
 3. Negate the statement which needs to prove (proof by contradiction)
 4. Draw resolution graph (unification).
-

Example :

- A. John likes all kind of food.
 - B. Apple and vegetable are food
 - C. Anything anyone eats and not killed is food.
 - D. Anil eats peanuts and still alive
 - E. Harry eats everything that Anil eats.
Prove by resolution that:
 - F. John likes peanuts.
-

Step-1: Conversion of Facts into FOL

A. $\forall x: \text{food}(x) \supseteq \text{likes}(\text{John}, x)$

B. $\text{food}(\text{Apple}) \wedge \text{food}(\text{vegetables})$

C. $\forall x \forall y: \text{eats}(x, y) \wedge \neg \text{killed}(x) \wedge \neg \text{food}(v)$

D. $\text{eats}(\text{Anil}, \text{Peanuts}) \wedge \text{alive}(\text{Anil})$

E. $\forall x: \text{eats}(\text{Anil}, x) \supset \text{eats}(\text{Harry}, x)$

F. $\forall x: \neg \text{killed}(x) \supset \text{alive}(x)$

G. $\forall x: \text{alive}(x) \supset \neg \text{killed}(x)$

H. $\text{likes}(\text{John}, \text{Peanuts})$



Added predicates

Step-2: Conversion of FOL into CNF

Eliminate all implication (\rightarrow) and rewrite

- A. $\forall x \neg \text{food}(x) \vee \text{likes}(\text{John}, x)$
 - B. $\text{food}(\text{Apple}) \wedge \text{food}(\text{vegetables})$
 - C. $\forall x \forall y \neg [\text{eats}(x, y) \wedge \neg \text{killed}(x)] \vee \text{food}(y)$
 - D. $\text{eats}(\text{Anil}, \text{Peanuts}) \wedge \text{alive}(\text{Anil})$
 - E. $\forall x \neg \text{eats}(\text{Anil}, x) \vee \text{eats}(\text{Harry}, x)$
 - F. $\forall x \neg [\neg \text{killed}(x)] \vee \text{alive}(x)$
 - G. $\forall x \neg \text{alive}(x) \vee \neg \text{killed}(x)$
 - H. $\text{likes}(\text{John}, \text{Peanuts})$.
-

Move negation (\neg) inwards and rewrite

- A. $\forall x \neg \text{food}(x) \vee \text{likes}(\text{John}, x)$
 - B. $\text{food}(\text{Apple}) \wedge \text{food}(\text{vegetables})$
 - C. $\forall x \forall y \neg \text{eats}(x, y) \vee \text{killed}(x) \vee \text{food}(y)$
 - D. $\text{eats}(\text{Anil}, \text{Peanuts}) \wedge \text{alive}(\text{Anil})$
 - E. $\forall x \neg \text{eats}(\text{Anil}, x) \vee \text{eats}(\text{Harry}, x)$
 - F. $\forall x \neg \text{killed}(x) \vee \text{alive}(x)$
 - G. $\forall x \neg \text{alive}(x) \vee \neg \text{killed}(x)$
 - H. $\text{likes}(\text{John}, \text{Peanuts})$.
-

Rename variables or standardize variables

- A. $\forall x \neg \text{food}(x) \vee \text{likes}(\text{John}, x)$
 - B. $\text{food}(\text{Apple}) \wedge \text{food}(\text{vegetables})$
 - C. $\forall y \forall z \neg \text{eats}(y, z) \vee \text{killed}(y) \vee \text{food}(z)$
 - D. $\text{eats}(\text{Anil}, \text{Peanuts}) \wedge \text{alive}(\text{Anil})$
 - E. $\forall w \neg \text{eats}(\text{Anil}, w) \vee \text{eats}(\text{Harry}, w)$
 - F. $\forall g \neg \text{killed}(g) \vee \text{alive}(g)$
 - G. $\forall k \neg \text{alive}(k) \vee \neg \text{killed}(k)$
 - H. $\text{likes}(\text{John}, \text{Peanuts})$.
-

-
- **Eliminate existential instantiation quantifier by elimination.** In this step, we will eliminate existential quantifier \exists , and this process is known as **Skolemization**. But in this example problem since there is no existential quantifier so all the statements will remain same in this step.
-

Drop Universal quantifiers

In this step we will drop all universal quantifier since all the statements are not implicitly quantified so we don't need it.

A. $\neg \text{food}(x) \vee \text{likes}(\text{John}, x)$

B. $\text{food}(\text{Apple})$

C. $\text{food}(\text{vegetables})$

D. $\neg \text{eats}(y, z) \vee \text{killed}(y) \vee \text{food}(z)$

E. $\text{eats}(\text{Anil}, \text{Peanuts})$

F. $\text{alive}(\text{Anil})$

G. $\neg \text{eats}(\text{Anil}, w) \vee \text{eats}(\text{Harry}, w)$

H. $\text{killed}(g) \vee \text{alive}(g)$

I. $\neg \text{alive}(k) \vee \neg \text{killed}(k)$

J. $\text{likes}(\text{John}, \text{Peanuts})$.

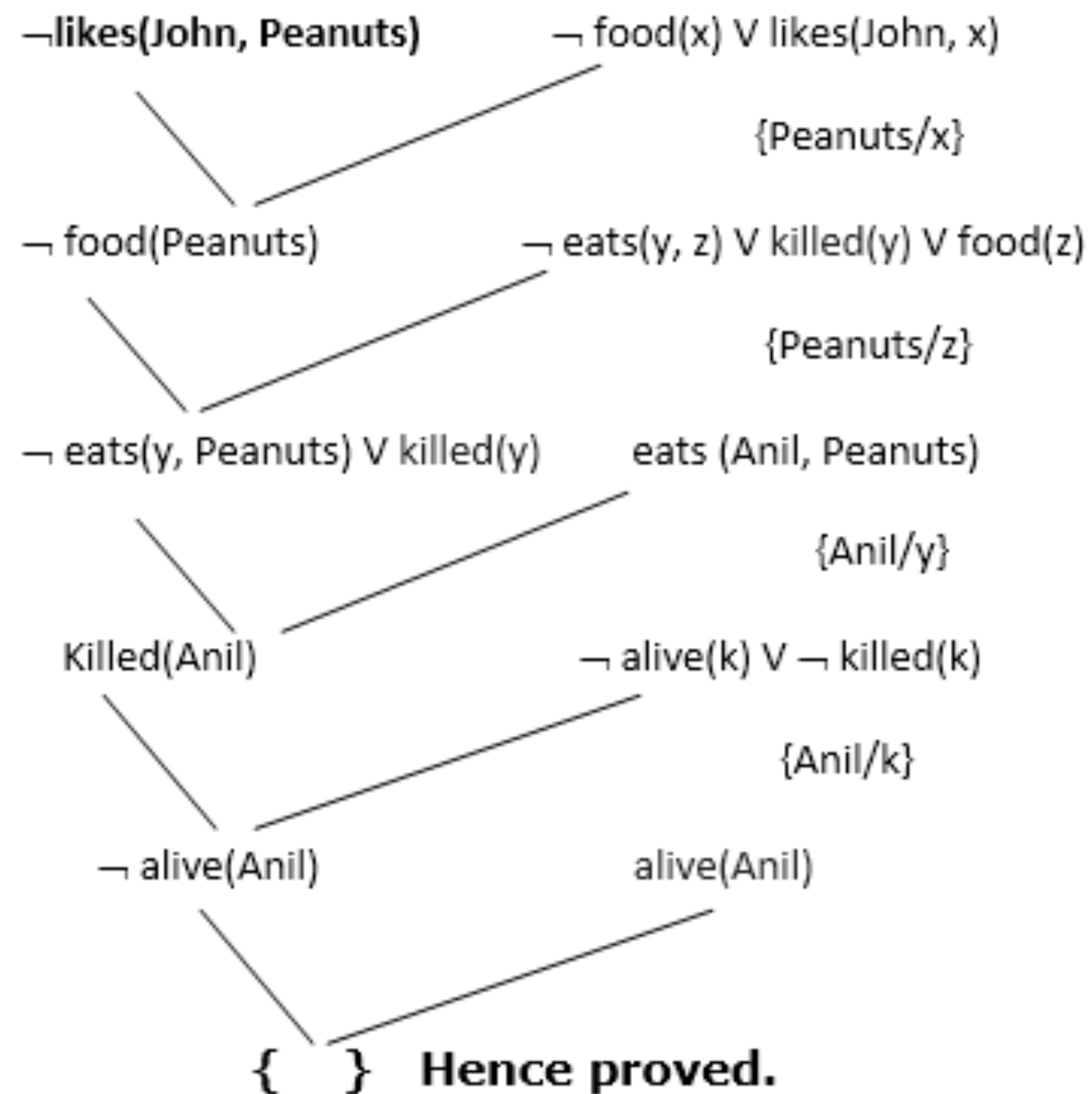
**"food(Apple) \wedge food(vegetables)" and
"eats (Anil, Peanuts) \wedge alive(Anil)"
can be written in two separate
statements.**

Distribute conjunction \wedge over disjunction \vee

This step will not make any change in this problem

Step-3: Negate the statement to be proved

In this statement, we will apply negation to the conclusion statements, which will be written as $\neg \text{likes}(\text{John}, \text{Peanuts})$



Hence the negation of the conclusion has been proved as a complete contradiction with the given set of statements.

Explanation of Resolution graph

- In the first step of resolution graph, $\neg \text{likes}(\text{John}, \text{Peanuts})$, and $\text{likes}(\text{John}, x)$ get resolved(canceled) by substitution of $\{\text{Peanuts}/x\}$, and we are left with $\neg \text{food}(\text{Peanuts})$
 - In the second step of the resolution graph, $\neg \text{food}(\text{Peanuts})$, and $\text{food}(z)$ get resolved (canceled) by substitution of $\{\text{Peanuts}/z\}$, and we are left with $\neg \text{eats}(y, \text{Peanuts}) \vee \text{killed}(y)$.
 - In the third step of the resolution graph, $\neg \text{eats}(y, \text{Peanuts})$ and $\text{eats}(\text{Anil}, \text{Peanuts})$ get resolved by substitution $\{\text{Anil}/y\}$, and we are left with $\text{Killed}(\text{Anil})$.
 - In the fourth step of the resolution graph, $\text{Killed}(\text{Anil})$ and $\neg \text{killed}(k)$ get resolve by substitution $\{\text{Anil}/k\}$, and we are left with $\neg \text{alive}(\text{Anil})$.
 - In the last step of the resolution graph $\neg \text{alive}(\text{Anil})$ and $\text{alive}(\text{Anil})$ get resolved.
-

Forward & Backward Chaining

Inference engine : The inference engine is the component of the intelligent system in artificial intelligence, which applies logical rules to the knowledge base to infer new information from known facts.

Definite clause: A clause which is a disjunction of literals with **exactly one positive literal** is known as a definite clause or strict horn clause.

Horn clause: A clause which is a disjunction of literals with **at most one positive literal** is known as horn clause. Hence all the definite clauses are horn clauses.

Example: $(\neg p \vee \neg q \vee k)$. It has only one positive literal k.

It is equivalent to $p \wedge q \rightarrow k$.

Forward Chaining

- Forward chaining is also known as a forward deduction or forward reasoning method.
 - Forward chaining is a form of reasoning which start with atomic sentences in the knowledge base and applies inference rules 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.
-

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.
 - Forward-chaining approach is also called as data-driven as we reach to the goal using available data.
 - Forward -chaining approach is commonly used in the expert system, such as CLIPS, business, and production rule systems.
-

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.
-

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) \wedge weapon(q) \wedge sells(p, q, r) \wedge hostile(r) \rightarrow Criminal(p) ... (1)

- Country A has some missiles. **?p Owns(A, p) \wedge 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)

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

?p Missiles(p) \wedge Owns(A, p) \rightarrow Sells(Robert, p, A) (4)

- Missiles are weapons.

Missile(p) \rightarrow Weapons(p) (5)

- Enemy of America is known as hostile.

Enemy(p, America) \rightarrow Hostile(p) (6)

- Country A is an enemy of America.

Enemy(A, America) (7)

- Robert is American

American(Robert). (8)

Forward chaining proof

- 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.

American (Robert)

Missile (T1)

Owns (A,T1)

Enemy (A, America)

Step-2:

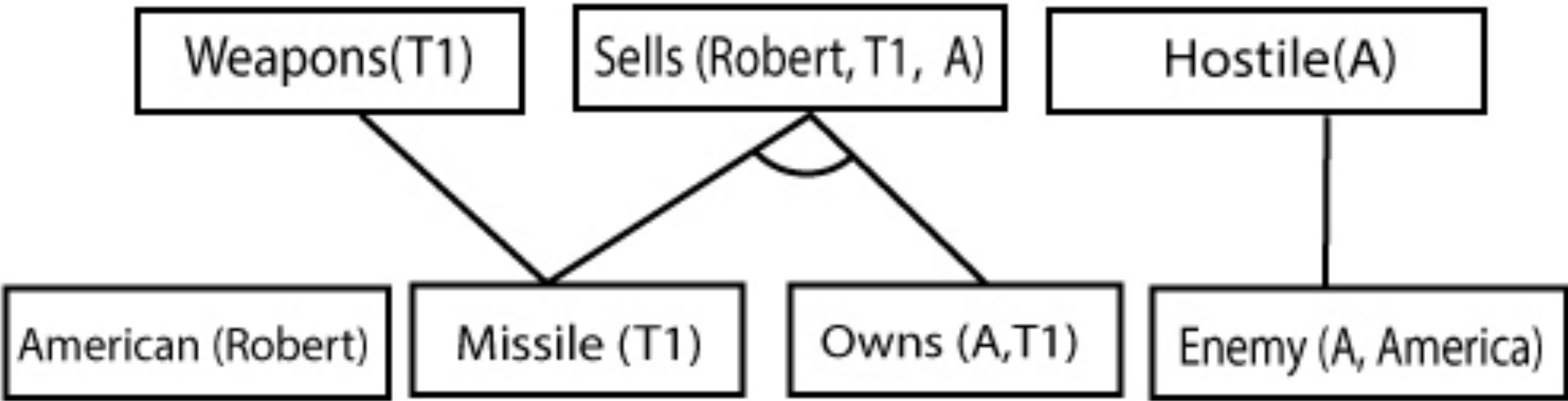
At the second step, we will see those facts which infer from available facts and with satisfied premises.

Rule-(1) does not satisfy premises, so it will not be added in the first iteration.

Rule-(2) and (3) are already added.

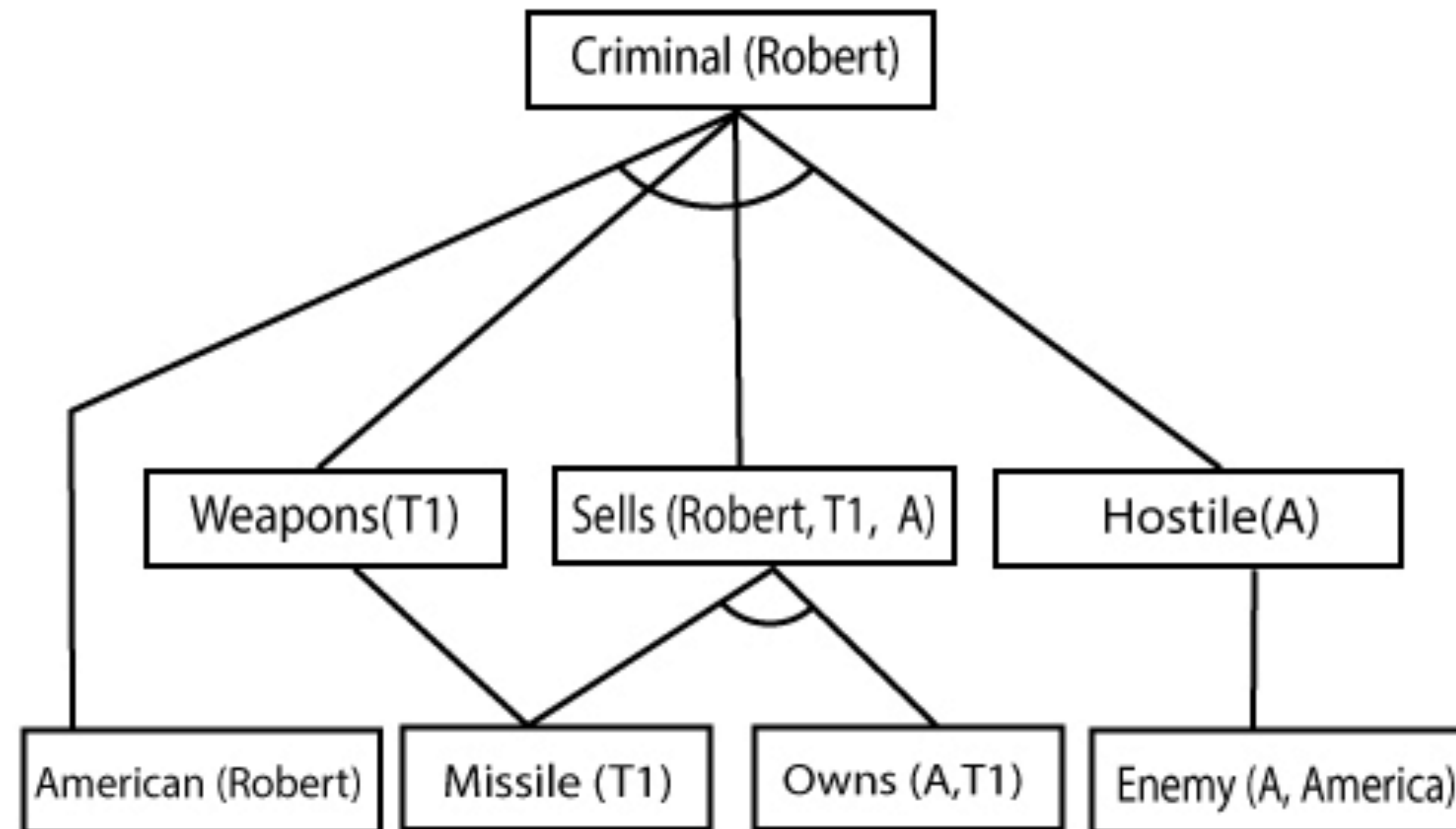
Rule-(4) satisfy with the substitution {p/T1}, **so Sells (Robert, T1, A)** is added, which infers from the conjunction of Rule (2) and (3).

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



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.



Hence it is proved that Robert is Criminal using forward chaining approach.

Backward Chaining

- Backward-chaining is also known as a backward deduction or backward reasoning method.
 - 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.
-

Properties :

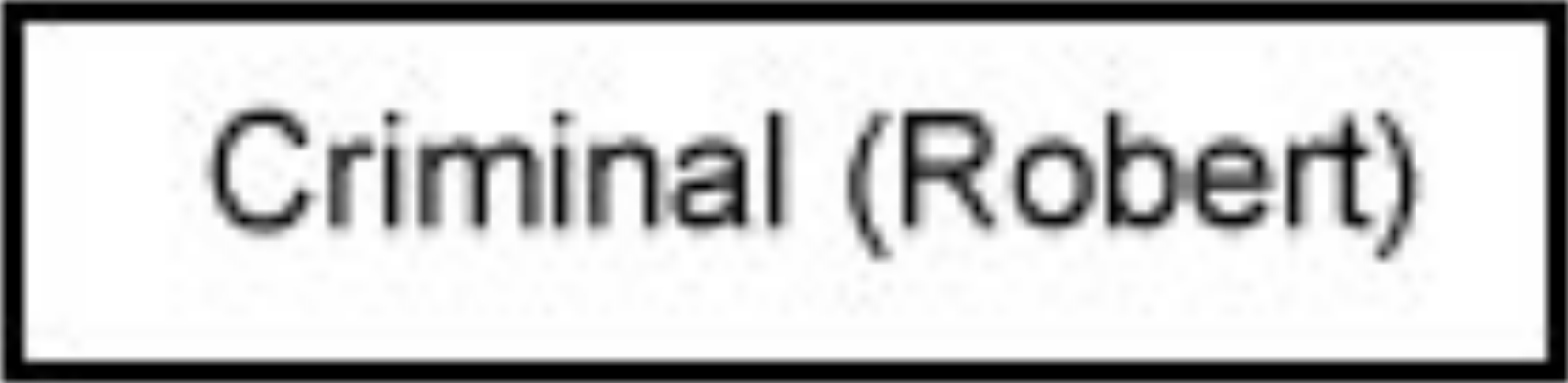
- It is known as a top-down approach.
 - Backward-chaining is based on modus ponens inference rule.
 - In backward chaining, the goal is broken into sub-goal or sub-goals to prove the facts true.
 - It is called a goal-driven approach, as a list of goals decides which rules are selected and used.
 - Backward -chaining algorithm is used in game theory, automated theorem proving tools, inference engines, proof assistants, and various AI applications.
 - The backward-chaining method mostly used a **depth-first search** strategy for proof.
-

Example :

- In backward-chaining, we will use the same above example, and will rewrite all the rules.
 - **American (p) \wedge weapon(q) \wedge sells (p, q, r) \wedge hostile(r) \rightarrow Criminal(p) ... (1)**
Owens(A, T1) (2)
 - **Missile(T1)**
 - **?p Missiles(p) \wedge Owens (A, p) \rightarrow Sells (Robert, p, A) (4)**
 - **Missile(p) \rightarrow Weapons (p) (5)**
 - **Enemy(p, America) \rightarrow Hostile(p) (6)**
 - **Enemy (A, America) (7)**
 - **American(Robert). (8)**
-

Backward-Chaining proof:

- 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.

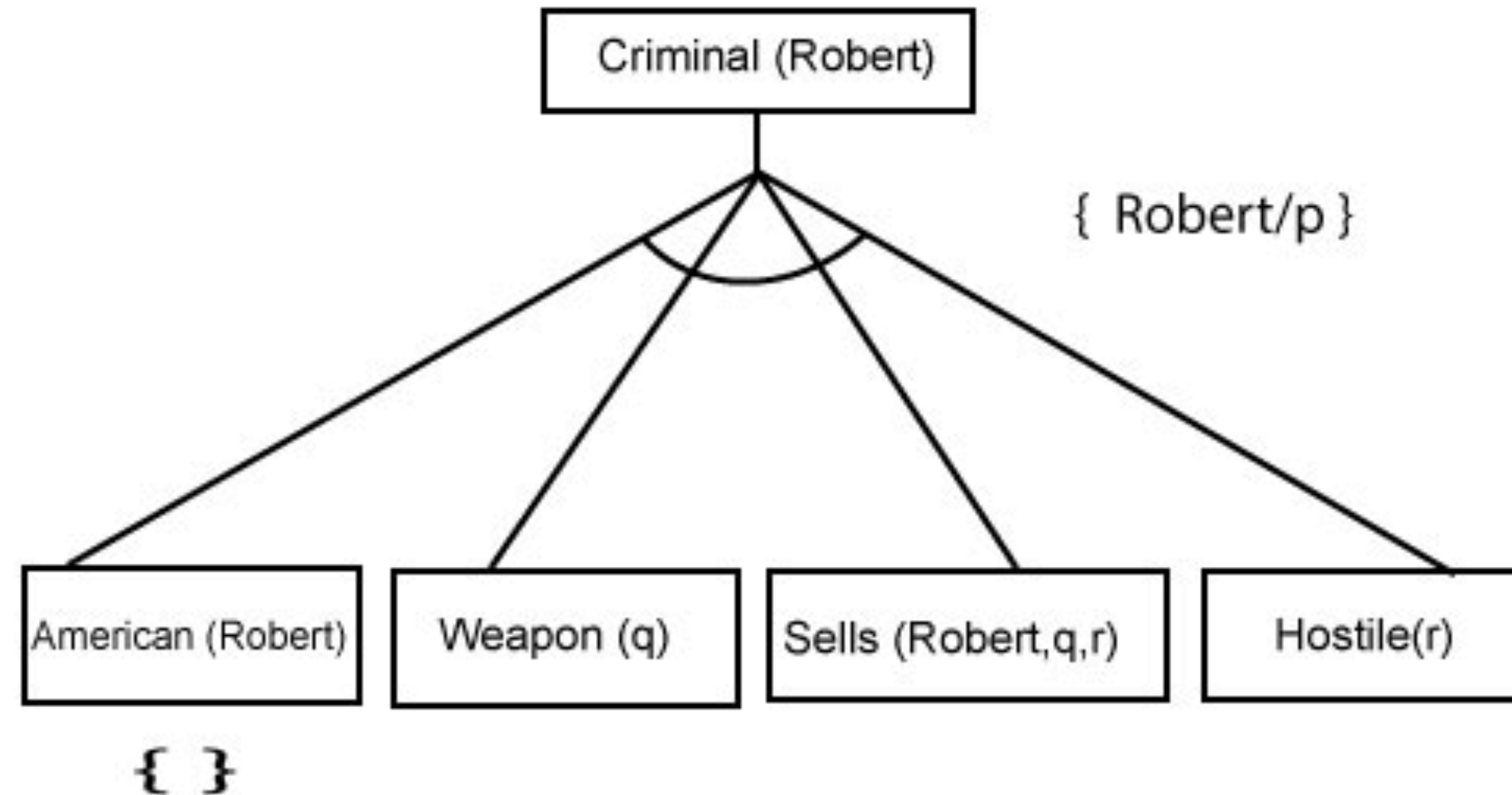


Criminal (Robert)

- Step-2:

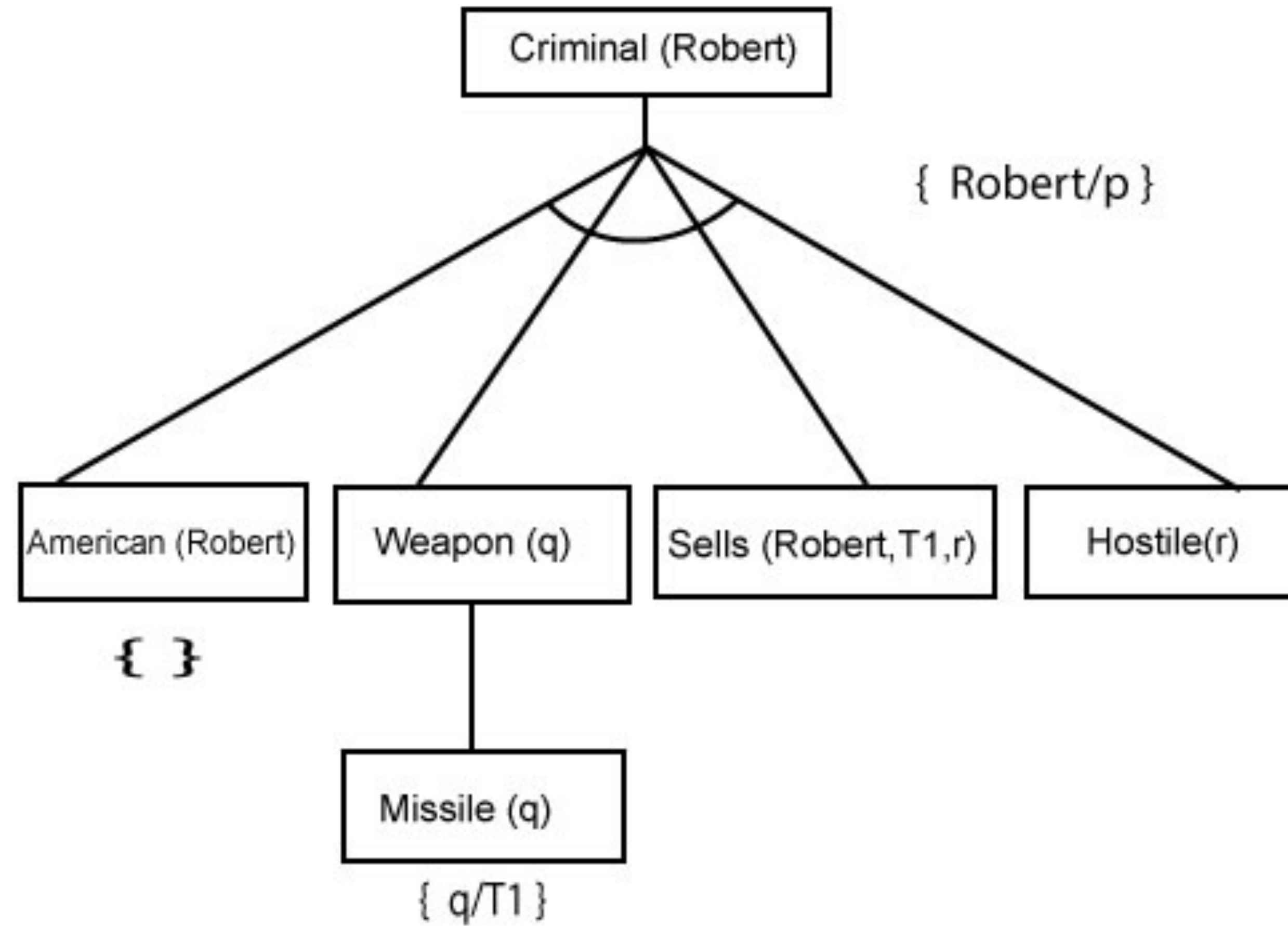
At the second step, we will infer other facts from goal fact which satisfies the rules. So as we can see in Rule-1, the goal predicate Criminal (Robert) is present with substitution {Robert/P}. So we will add all the conjunctive facts below the first level and will replace p with Robert.

- Here we can see American (Robert) is a fact, so it is proved here.



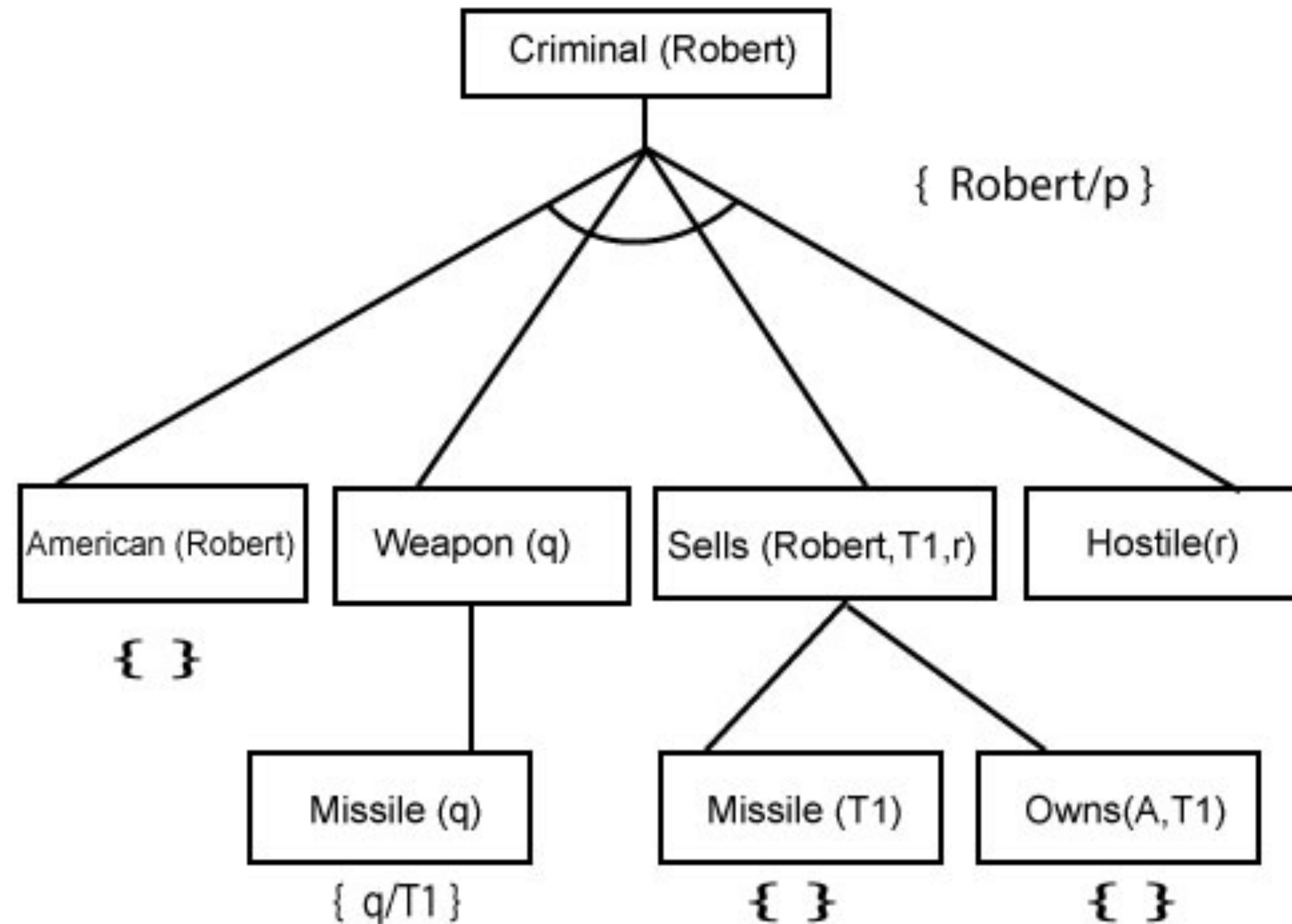
- **Step-3:**

At 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.



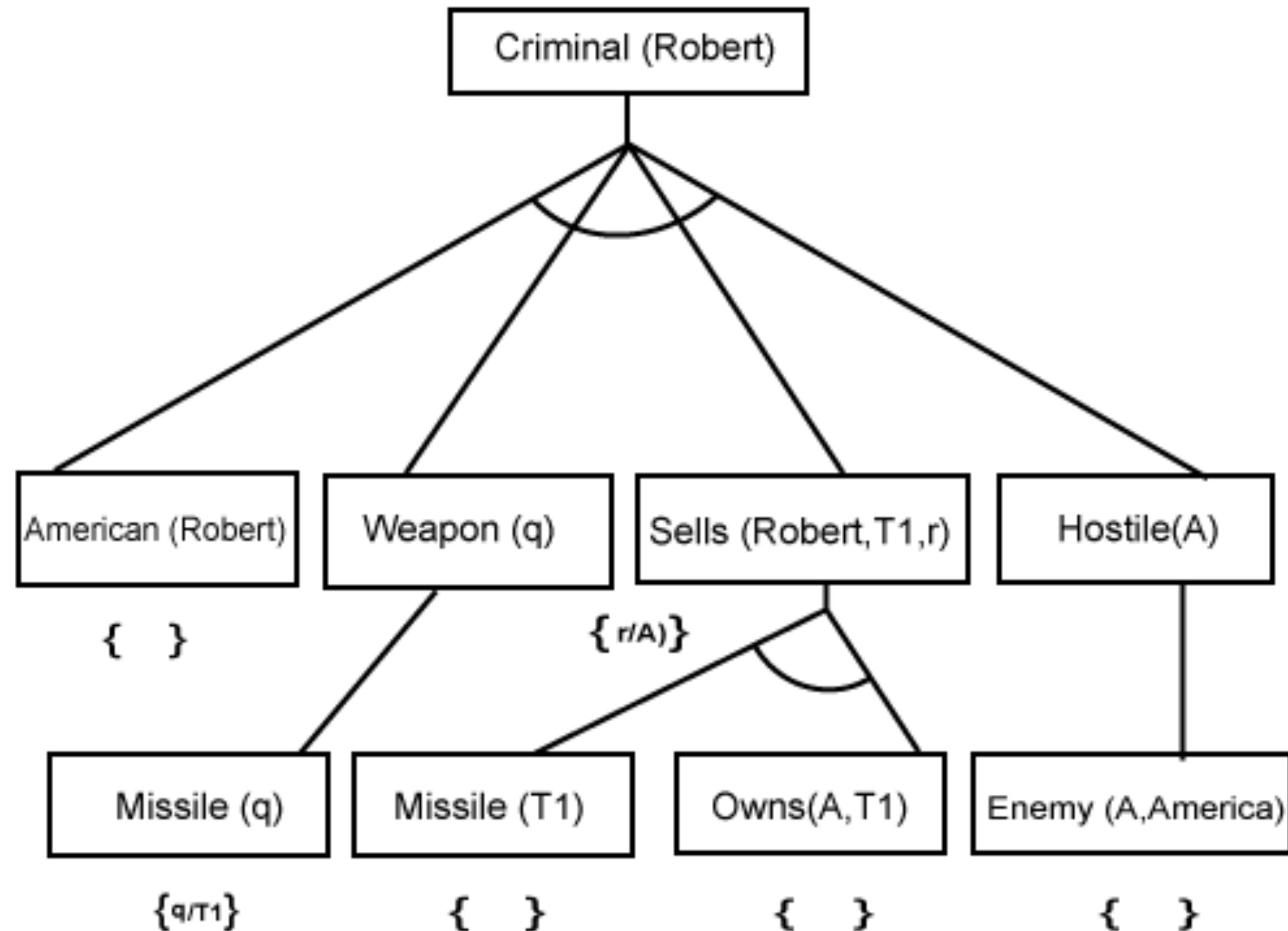
- **Step-4:**

At step-4, we can infer facts Missile(T1) and Owns(A, T1) from 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.



- **Step-5:**

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 vs Backward

-
- Forward chaining as the name suggests, start from the known facts and move forward by applying inference rules to extract more data, and it continues until it reaches to the goal, whereas backward chaining starts from the goal, move backward by using inference rules to determine the facts that satisfy the goal.
 - Forward chaining is called a **data-driven** inference technique, whereas backward chaining is called a **goal-driven** inference technique.
 - Forward chaining is known as the **down-up** approach, whereas backward chaining is known as a **top-down** approach.
 - Forward chaining uses **breadth-first search** strategy, whereas backward chaining uses **depth-first search** strategy.
-

-
- Forward and backward chaining both applies **Modus ponens** inference rule.
 - Forward chaining can be used for tasks such as planning, design process monitoring, diagnosis, and classification, whereas backward chaining can be used for classification and diagnosis tasks.
 - Forward chaining can be like an exhaustive search, whereas backward chaining tries to avoid the unnecessary path of reasoning.
 - In forward-chaining there can be various ASK questions from the knowledge base, whereas in backward chaining there can be fewer ASK questions.
 - Forward chaining is slow as it checks for all the rules, whereas backward chaining is fast as it checks few required rules only.
-

1.	Forward chaining starts from known facts and applies inference rule to extract more data unit it reaches to the goal.	Backward chaining starts from the goal and works backward through inference rules to find the required facts that support the goal.
2.	It is a bottom-up approach	It is a top-down approach
3.	Forward chaining is known as data-driven inference technique as we reach to the goal using the available data.	Backward chaining is known as goal-driven technique as we start from the goal and divide into sub-goal to extract the facts.
4.	Forward chaining reasoning applies a breadth-first search strategy.	Backward chaining reasoning applies a depth-first search strategy.
5.	Forward chaining tests for all the available rules	Backward chaining only tests for few required rules.
6.	Forward chaining is suitable for the planning, monitoring, control, and interpretation application.	Backward chaining is suitable for diagnostic, prescription, and debugging application.

7.	Forward chaining can generate an infinite number of possible conclusions.	Backward chaining generates a finite number of possible conclusions.
8.	It operates in the forward direction.	It operates in the backward direction.
9.	Forward chaining is aimed for any conclusion.	Backward chaining is only aimed for the required data.
