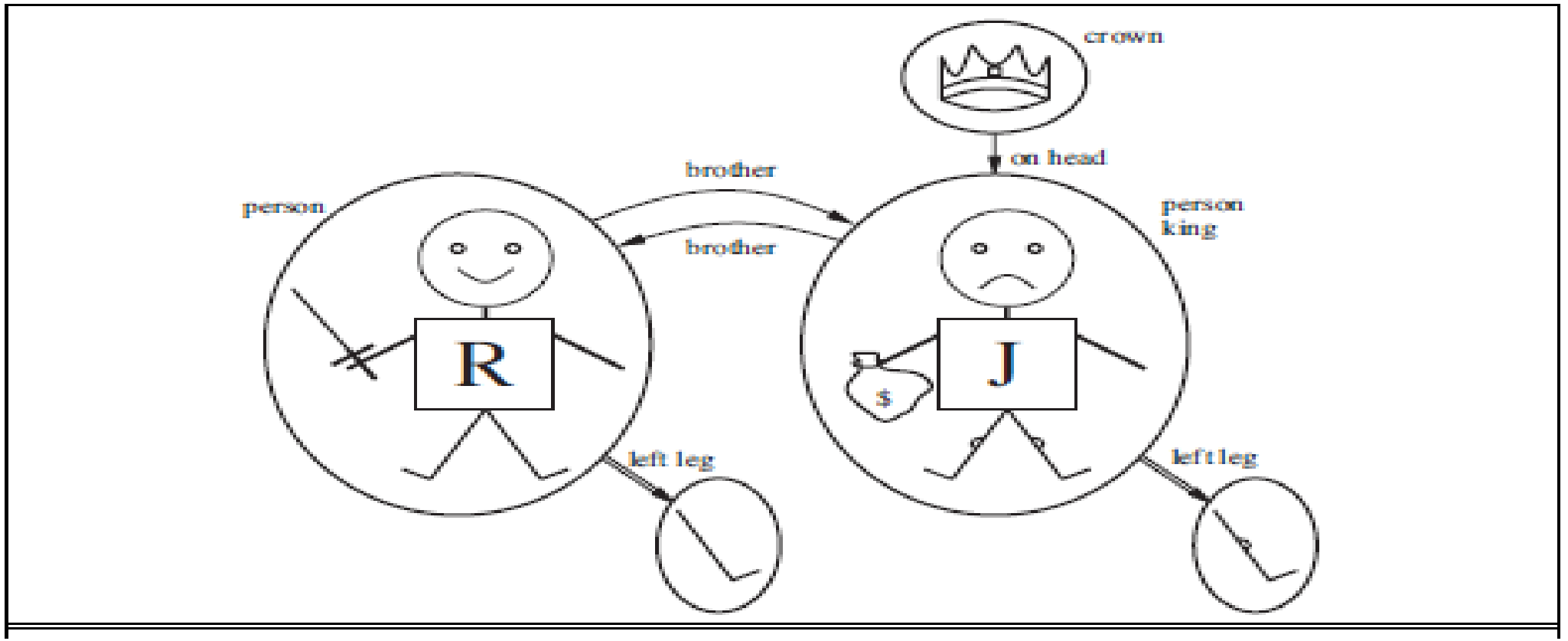
First Order Logic

Language	Ontological Commitment (What exists in the world)	Epistemological Commitment (What an agent believes about facts)
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief $\in [0, 1]$
Fuzzy logic	facts with degree of truth $\in [0, 1]$	known interval value

Formal languages and their ontological and epistemological commitments.

FOL



A model containing five objects, two binary relations, three unary relations (indicated by labels on the objects), and one unary function, left-leg.

FOL

Sentence → AtomicSentence | ComplexSentence

AtomicSentence → Predicate | Predicate(Term, . . .) | Term = Term

ComplexSentence → (Sentence) | [Sentence]

| ¬Sentence

| Sentence ∧ Sentence

| Sentence ∨ Sentence

| Sentence ⇒ Sentence

| Sentence ⇔ Sentence

| Quantifier Variable, . . . Sentence

FOL

Term → Function(Term, . . .)

| Constant

| Variable

Quantifier → ∀ | ∃

Constant → A | X1 | John | · · ·

Variable → a | x | s | ⋯

Predicate → True | False | After | Loves | Raining | ⋯

Function → Mother | LeftLeg | · · ·

OPERATOR PRECEDENCE : ¬,=, ∧, ∨,⇒,⇔

FOL

Atomic sentences

LeftLeg(John) refers to King John's left leg. - Function

Brother (Richard , John). - Predicate

Married(Father (Richard),Mother (John)) - Complex terms as arguments

An atomic sentence is true in a given model if the relation referred to by the predicate symbol holds among the objects referred to by the arguments.

Complex Sentences

¬Brother (LeftLeg(Richard), John)

Brother (Richard , John) ∧ Brother (John,Richard)

King(Richard) ∨ King(John)

¬King(Richard) ⇒ King(John)

FOL

$\forall$ x King(x) $\Rightarrow$ Person(x)

x $\rightarrow$ Richard the Lionheart,

x $\rightarrow$ King John,

x $\rightarrow$ Richard's left leg,

x $\rightarrow$ John's left leg,

x $\rightarrow$ the crown.

Richard the Lionheart is a king $\Rightarrow$ Richard the Lionheart is a person.

King John is a king $\Rightarrow$ King John is a person.

Richard's left leg is a king $\Rightarrow$ Richard's left leg is a person.

John's left leg is a king $\Rightarrow$ John's left leg is a person.

The crown is a king $\Rightarrow$ the crown is a person.

$\forall$ x $\forall$ y Brother (x, y) $\Rightarrow$ Sibling(x, y) .

Implication is true when premise is True - intended interpretation

FOL

wrong

$\forall x \; King(x) \land Person(x)$

Richard the Lionheart is a king ∧ Richard the Lionheart is a person,

King John is a king ∧ King John is a person,

Richard's left leg is a king ∧ Richard's left leg is a person,

Implication is true whenever premise is False – regardless of truth of conclusion - extended interpretation

FOL

$\exists$ x Crown(x) $\wedge$ OnHead(x, John)

$\exists$x is pronounced “There exists an x such that . . .” or “For some x . . .”

Richard the Lionheart is a crown $\wedge$ Richard the Lionheart is on John’s head;

King John is a crown $\wedge$ King John is on John’s head;

Richard’s left leg is a crown $\wedge$ Richard’s left leg is on John’s head;

John’s left leg is a crown $\wedge$ John’s left leg is on John’s head;

The crown is a crown $\wedge$ the crown is on John’s head.

Just as$\Rightarrow$appears to be the natural connective to use with $\forall$, $\wedge$ is the natural connective to use with $\exists$.

FOL - nested quantifiers

$\forall x \forall y$ Brother $(x, y) \Rightarrow$ Sibling(x, y)

$\forall x, y$ Sibling$(x, y) \Leftrightarrow$ Sibling(y, x)

$\forall x \exists y$ Loves(x, y) . Everybody loves somebody

$\exists y \forall x$ Loves(x, y) . There is someone loved by everyone

$\forall x$ (Crown$(x) \vee (\exists x$ Brother (Richard, $x)))$ variable applies to innermost quantifier

Quantifiers

$\forall x \neg Likes(x, cabbage)$ is equivalent to $\neg\exists x\ Likes(x, cabbage)$. Everyone dislikes cabbage

$\forall x\ Likes(x, IceCream)$ is equivalent to $\neg\exists x \neg Likes(x, IceCream)$.

$\forall x \neg P \equiv \neg\exists x\ P$

$\neg\forall x\ P \equiv \exists x \neg P$

$\forall x\ P \equiv \neg\exists x \neg P$

$\exists x\ P \equiv \neg\forall x \neg P$

$\neg(P \vee Q) \equiv \neg P \wedge \neg Q$

$\neg(P \wedge Q) \equiv \neg P \vee \neg Q$

$P \wedge Q \equiv \neg(\neg P \vee \neg Q)$

$P \vee Q \equiv \neg(\neg P \wedge \neg Q)$.

FOL - Equality

Father (John)=Henry

Richard has two brothers, John and Geoffrey.

$\exists$ x, y Brother (x,Richard) $\wedge$ Brother (y,Richard) ?

$\exists$ x, y Brother (x,Richard) $\wedge$ Brother (y,Richard) $\wedge\neg$(x=y) .

Brother (John,Richard) $\wedge$ Brother (Geoffrey,Richard) ?

Brother (John,Richard) $\wedge$ Brother (Geoffrey,Richard) $\wedge$ John $\neq$ Geoffrey

$\wedge \forall$x Brother (x,Richard) $\Rightarrow$ (x=John $\vee$ x=Geoffrey) .

Database Semantics

Database Semantics

UNIQUE- NAMES assumption - every constant symbol refer to a distinct object

CLOSED-WORLD assumption - atomic sentences not known to be true are in fact false

Domain closure – Each model contains no more domain elements than those named by the constant symbols.

FOL assertions and queries

TELL(KB, King(John)) .

TELL(KB, Person(Richard)) .

TELL(KB, $\forall$ x King(x) $\Rightarrow$ Peson(x))

ASK(KB, King(John))

ASK(KB, Person(John))

ASK(KB, $\exists$ x Person(x)) . Returns T

ASKVARS(KB, Person(x)) Returns stream of answers

FOL - Kinship Domain

one's mother is one's female parent:

$\forall$ m, c Mother (c)=m $\Leftrightarrow$ Female(m) $\wedge$ Parent(m, c) .

One's husband is one's male spouse:

$\forall$ w, h Husband(h,w) $\Leftrightarrow$ Male(h) $\wedge$ Spouse(h,w) .

Male and female are disjoint categories:

$\forall$ x Male(x) $\Leftrightarrow$ $\neg$Female(x) .

Parent and child are inverse relations:

$\forall$ p, c Parent(p, c) $\Leftrightarrow$ Child (c, p) .

A grandparent is a parent of one's parent:

$\forall$ g, c Grandparent (g, c) $\Leftrightarrow$ $\exists$p Parent(g, p) $\wedge$ Parent(p, c) .

A sibling is another child of one's parents:

$\forall$ x, y Sibling(x, y) $\Leftrightarrow$ x $\neq$ y $\wedge$ $\exists$p Parent(p, x) $\wedge$ Parent(p, y) .

Wumpus World

Input – percept sentence with timestep

Percept ([Stench, Breeze, Glitter , None, None], 5) .

Actions in the wumpus world can be represented by logical terms:

Turn(Right), Turn(Left), Forward , Shoot , Grab, Climb .

To determine which is best, the agent program executes the query

ASKVARS(∃ a BestAction(a, 5))

The raw percept data implies certain facts about the current state (perception rules)

∀ t, s, g, m, c Percept ([s, Breeze, g,m, c], t) ⇒ Breeze(t) ,

∀ t, s, b, m, c Percept ([s, b, Glitter,m, c], t) ⇒ Glitter (t) ,

Reflex behavior can also be implemented by quantified implication sentences.

∀ t Glitter (t) ⇒ BestAction(Grab, t)

Wumpus world

Adjacency of any two squares :

$\forall x, y, a, b$ Adjacent $([x, y], [a, b]) \Leftrightarrow (x = a \wedge (y = b - 1 \vee y = b + 1)) \vee (y = b \wedge (x = a - 1 \vee x = a + 1))$.

Agent is at square s at time t.

At(Agent, s, t)

Given its current location, the agent can infer properties of the square from properties of its current percept.

$\forall s, t$ At(Agent, s, t) $\wedge$ Breeze(t) $\Rightarrow$ Breezy(s)

agent can deduce where the pits are with one axiom

$\forall s$ Breezy(s) $\Leftrightarrow \exists r$ Adjacent (r, s) $\wedge$ Pit(r)

Knowledge Engineering

1. Identify the task.
2. Assemble the relevant knowledge.
3. Decide on a vocabulary of predicates, functions, and constants.
4. Encode general knowledge about the domain.
5. Encode a description of the specific problem instance.
6. Pose queries to the inference procedure and get answers.
7. Debug the knowledge base.

Example

1. Leena is a professor
2. All professors are people.
3. Frank is the dean.
4. Deans are professors.
5. All professors consider the dean a friend or don't know him.
6. Everyone is a friend of someone.
7. People only criticize people that are not their friends.
8. Leena criticized Frank.

KB

prof(leena)

∀ x (prof(x) => person(x))

dean(frank)

∀ x (dean(x) => prof(x))

∀ x (∀ y (prof(x) ∧ dean(y) => friend(y, x) ∨ ¬ knows(x, y)))

∀ x (∃ y (friend(y, x)))

∀ x (∀ y (person(x) ∧ person(y) ∧ criticize (x,y) => ¬ friend (y,x)))

criticize(leena,frank)

Question: Is Frank no friend of Leena?

¬ friend (frank,leena)

Unit resolution inference rule

each *l* is a literal, *l*i and *m* are complementary

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k},$$

Unit resolution inference rule takes a clause - a disjunction of literals

and a literal as input

and produces a new clause

Full resolution rule: *li* and *mj* are complementary

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n},$$

Ex:

$$\frac{P_{1,1} \vee P_{3,1}, \qquad \neg P_{1,1} \vee \neg P_{2,2}}{P_{3,1} \vee \neg P_{2,2}}$$

Factoring: removal of multiple copies of literals: by resolving (A V B) with (A V ¬ B) , we get (A V A), reduced to A

Resolution rule forms basis for Sound , complete inference procedures

A resolution-based theorem prover can, for any sentences α and β in propositional logic, decide whether α |= β.

Resolution

Resolution Rule of Inference

$$\frac{P_{1,1} \vee P_{3,1}, \qquad \neg P_{1,1} \vee \neg P_{2,2}}{P_{3,1} \vee \neg P_{2,2}}$$

Algorithm: Resolution Proof

Negate the theorem to be proved, and add the result to the knowledge base.

Bring knowledge base into conjunctive normal form (CNF) -

CNF: conjunctions of disjunctions. Each disjunction is called a clause.

Until there is no resolvable pair of clauses,

Find resolvable clauses and resolve them.

Add the results of resolution to the knowledge base.

If NIL (empty clause) is produced, stop and report that the (original) theorem is true.

Report that the (original) theorem is false.

Resolution

Resolution Example: Propositional Logic

- To prove: ¬ P
- Transform Knowledge Base into CNF
- Proof

1. ¬P ∨Q	Sentence 1
2. ¬ Q ∨R	Sentence 2
3. ¬R	Sentence 3
4. P	Assume opposite
5. Q	Resolve 4 and 1
6. R	Resolve 5 and 2
7. nil	Resolve 6 with 3

More examples on FOL

Atomic sentences: Predicate | Predicate (Term, …) | Term = Term

Term = function(Term, …)

Function appears only as an attribute in a predicate.

Predicate has a truth value for the domains of attributes, when instantiated to object values.

1. Everyone loves its mother ∀x ∃y Mother(x,y) ∧ Loves(x,y) : Mother, Loves :Predicates

   ∀ x Loves(x, Mother(x)) : Mother : function

2. Not all students take both History and Biology

   predicates: student(x) : x is a student

   Takes (x, y) : subject x is taken by y

   constants: History, Biology

   ¬ [∀ x student(x) => Takes(History,x) ∧ Takes(Biology,x)]

   ∃x student(x) ∧ [¬ Takes(History,x) ∨ ¬ Takes(Biology,x)]

- ∀x ∀y [student(x) ∧ Subject (y) ∧ All_Attendance_Ok(x) ∧ Sub_Attendance_Ok(x, y) => End_Sem_Exam(x, y)

More examples on FOL

3. Only one student Failed in History

predicate: Failed(x,y) : student y failed in subject x

$\exists x$ student(x) $\wedge$ Failed(History, x) : There is somebody who failed in History

$\exists x$ [student(x) $\wedge$ Failed(History, x) $\wedge \forall y$ [$\neg$ (x=y) $\wedge$ student(y) => $\neg$ Failed (History, y)]]

4. Only one student failed in both History and Biology

$\exists x$ [student(x) $\wedge$ Failed(History,x) $\wedge$ Failed(Biology,x) $\wedge$

$\forall y$[$\neg$ (x=y) $\wedge$ student(y) => $\neg$ Failed(History, y) $\wedge$ $\neg$ Failed(Biology,y)]]

5. The best score in History is better than best score in Biology

function: score(subject, student)

Predicate: Greater(x,y) : x>y

More examples on FOL

5. The best score in History is better than best score in Biology

function: Score(subject, student)

Predicate: Greater(x,y) : x>y

$\forall x$ [student(x) $\wedge$ Takes(Biology,x) => $\exists y$ student(y) $\wedge$ Takes(History,y) $\wedge$
Greater(Score(History,y), Score(Biology,x))]

6. No Person likes a Professor unless the Professor is Smart

Predicate: Smart (x)

Likes(x, y) : y likes x

$\forall x$ Professor(x) $\wedge$ $\neg$Smart(x) => $\forall y$ $\neg$Likes(x,y)

More examples on FOL

Russels Paradox:

There is a single barber in Town. Those and those who do not shave themselves are shaved by the barber. Who shaves barber?

Predicates: Barber(x), Shaves(x,y) : y shaves x

$\exists x [Barber(x) \wedge \forall y \neg(x=y) \Rightarrow \neg Barber(y)]$:There is a single barber in town

$\forall x [\neg Shaves(x,x) \Leftrightarrow Shaves(x,y) \wedge Barber(y)]$

Homework:

Politicians can fool some of the persons all the time, and they can fool all the people some of the time, but can not fool all the people all the time.

Inference Rules for Quantifiers

Universal Quantifier ∀

KB: axiom: All greedy kings are evil

Predicates: King(x), Greedy (x), Evil (x)

Sentence: ∀ x King (x) ∧ Greedy (x) => Evil (x)

King(John) ∧ Greedy(John) ⇒ Evil(John) : {x/John}

King(Richard) ∧ Greedy(Richard) ⇒ Evil(Richard) : {x/Richard}

King(Father (John)) ∧ Greedy(Father (John)) ⇒ Evil(Father (John)) : {x/Father(John)}

Universal Instantiation (UI) : we can infer any sentence obtained by substituting a ground term (a term without variables)

$$\frac{\forall v \quad \alpha}{\text{SUBST}(\{v/g\}, \alpha)}$$

For any variable v and ground term g. SUBST (θ, α) denotes the result of applying the substitution θ to the sentence α

Inference Rules for Quantifiers

Existential Instantiation:

for any sentence α, variable v, and constant symbol k that does not appear elsewhere in the knowledge base,

$$\frac{\exists\, v \quad \alpha}{\text{SUBST}(\{v/k\}, \alpha)}$$

Variable v is replaced by a new constant symbol, k

From the sentence,

$$\exists\, x\ \text{Crown}(x) \wedge \text{OnHead}(x, \text{John})$$

we can infer the sentence

$$\text{Crown}(C1) \wedge \text{OnHead}(C1, \text{John})$$

as long as C1 does not appear elsewhere in the knowledge base.

New name for the object is Skolem Constant.

Existential Instantiation is a special case of a more general process called skolemization

Inference Rules for Quantifiers

Universal Instantiation can be applied many times to produce many different Consequences.

Existential Instantiation can be applied once, and then the existentially quantified sentence can be discarded.

Inference Rules for Quantifiers

Reduction to propositional inference

our knowledge base contains just the sentences

$\forall x \; King(x) \wedge Greedy(x) \Rightarrow Evil(x)$

King(John)

Greedy(John)

Brother (Richard, John)

apply UI to the first sentence using all possible ground-term substitutions from the vocabulary of the knowledge base

{x/John} and {x/Richard }.

We obtain

$King(John) \wedge Greedy(John) \Rightarrow Evil(John)$

$King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$

universally quantified sentence is now discarded.

All ground terms are similar to PL Symbols

Use PL inference to obtain conclusion.

Inference Rules for Quantifiers

Problem with FOL -> PL and PL inference

Function Father(Father(Father……Father(John)…….))

Possible ground term substitutions are infinity

Inference with infinitely large sentences is difficult in PL

Herbrand Theorem:

If a sentence is entailed by the original in FOL, there is a proof involving just a finite subset of PL KB.

We can find the subset by generating all the instantiations with constant symbols (Richard, John), then all the terms of depth1, (Father (Richard)), then depth 2, …. Until we are able to construct a PL proof of the entailed sentence

Proof with FOL → PL → PL inference is complete : any entailed sentence can be proved.

Can not say about non-entailment : if sentence in FOL is not entailed, we may end up nesting deeper and deeper – infinite loop

Entailment in FOL is semi-deductible

Algorithms exist that say yes to every entailed sentence, but no algorithm exists that also says no to every non-entailed sentence

Unification and Lifting

FOL → Pl inefficient

FOL inference rule:

$\forall$ x King(x) $\wedge$ Greedy(x) $\Rightarrow$ Evil(x)

King(John)

Greedy(John)

{x/John}

The inference that John is evil—that is, that {x/John} solves the query Evil(x)

If there is some substitution θ that makes each of the conjuncts of the premise of the implication identical to sentences already in the knowledge base, then we can assert the conclusion of the implication, after applying θ.

θ ={x/John}

Unification and Lifting

KB:

King(x)

King(John)

∀ y Greedy(y)

Inference should still work

we would still like to be able to conclude that Evil(John), because we know that John is a king (given) and John is greedy (because everyone is greedy)

find a substitution both for the variables in the implication sentence and for the variables in the sentences that are in the knowledge base.

applying the substitution

{x/John, y/John}

to the implication premises

King(x) and Greedy(x)

and the knowledge-base sentences

King(John) and Greedy(y)

will make them identical.

Unification and Lifting

Inference rule: Generalized Modus Ponens

For atomic sentences pi, pi′ , and q, where there is a substitution θ, such that

SUBST(θ, pi′)= SUBST(θ, pi), for all i,

$$\frac{p1', p2', \ldots, pn', \quad (p1 \wedge p2 \wedge \ldots \wedge pn \Rightarrow q)}{SUBST(\theta, q)}$$

.

There are n+1 premises to this rule: the n atomic sentences pi′ and the one implication. The conclusion is the result of applying the substitution θ to the consequent q. For our example:

P1′ is King(John)	p1 is King(x)
P2′ is Greedy(y)	p2 is Greedy(x)
θ is {x/John, y/John}	q is Evil(x)

SUBST(θ, q) is Evil(John) .

Unification and Lifting

Generalized Modus Ponens is a sound inference rule.

p |= SUBST(θ, p) holds by Universal Instantiation. It holds in particular for a θ that satisfies the conditions of the Generalized Modus Ponens rule.

Thus, from p1′ , . . . , pn′ we can infer

SUBST(θ, p1′∧ . . . ∧ SUBST(θ, pn′)

and from the implication

p1 ∧ . . . ∧ pn ⇒ q we can infer

SUBST(θ, p1) ∧ . . . ∧ SUBST(θ, pn) ⇒ SUBST(θ, q) .

Now, θ in Generalized Modus Ponens is defined so that

SUBST(θ, pi′)= SUBST(θ, pi), for all i;

Therefore the first of these two sentences matches the premise of the second exactly.

Hence, SUBST(θ, q) follows by Modus Ponens.

Generalized Modus Ponens is a lifted version of Modus Ponens—it raises Modus Ponens from ground (variable-free) propositional logic to first-order logic.

Unification and Lifting

Unification:

process of finding substitutions that make different logical expressions look identical.

UNIFY(p, q) = θ where SUBST(θ, p) = SUBST(θ, q) .

Example: Query: AskVars(Knows(John, x)): whom does John know?

Answer : find all sentences in the knowledge base that unify with Knows(John, x).

UNIFY (Knows(John, x), Knows(John, Jane)) = {x/Jane}

UNIFY (Knows(John, x), Knows(y, Bill)) = {x/Bill, y/John}

UNIFY (Knows(John, x), Knows(y,Mother (y))) = {y/John, x/Mother (John)}

UNIFY (Knows(John, x), Knows(x, Elizabeth)) = fail .

(x cannot take on the values John and Elizabeth at the same time.)

The problem can be avoided by **standardizing apart** one of the two sentences being unified, which means renaming its variables to avoid name clashes.

UNIFY(Knows(John, x), Knows(x20, Elizabeth)) = {x/Elizabeth, x20/John}

Unification and Lifting

UNIFY should return a substitution that makes the two arguments look the same. But there could be more than one such unifier.

example,

UNIFY(Knows(John, x), Knows(y, z)) could return

{y/John, x/z} or

{y/John, x/John, z/John}.

The first unifier gives Knows(John, z) as the result of unification,

The second gives Knows(John, John).

The second result could be obtained from the first by an additional substitution {z/John};

first unifier is more general than the second, because it places fewer restrictions on the values of the variables.

For every unifiable pair of expressions, there is a single most general unifier (

MGU) that is unique up to renaming and substitution of variables.

The process: recursively explore the two expressions simultaneously “side by side,” building up

a unifier along the way, but failing if two corresponding points in the structures do not match.

FOL Definitive clauses

FOL Definitive clauses are disjunctions of literals of which exactly one is positive.

A definite clause either is atomic or is an implication whose antecedent is a conjunction of positive literals and whose consequent is a single positive literal.

$King(x) \land Greedy(x) \Rightarrow Evil(x)$.

King(John) .

Greedy(y) .

Unlike propositional literals, first-order literals can include variables, in which case those variables are assumed to be universally quantified.

Example

The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

Represent the Knowledge Base in First order Logic and

prove that West is a criminal,

Using

Forward chaining

Backward chaining

Resolution

Example

The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

prove that West is a criminal.

it is a crime for an American to sell weapons to hostile nations

American(x) ∧Weapon(y) ∧ Sells(x, y, z) ∧ Hostile(z) ⇒ Criminal (x) 1

Nono has some missiles.

∃ x Owns(Nono, x)∧Missile(x) transformed into two definite clauses by Existential Instantiation, introducing a new constant M1:

Owns(Nono,M1) 2

Missile(M1) 3

All of its missiles were sold to it by Colonel West

Missile(x) ∧ Owns(Nono, x) ⇒ Sells(West, x, Nono) 4

We will also need to know that missiles are weapons:

Missile(x) ⇒ Weapon(x) 5

Example

and we must know that an enemy of America counts as hostile :

Enemy(x, America) ⇒ Hostile(x) 6

West, who is American :

American(West) . 7

The country Nono, an enemy of America :

Enemy(Nono, America) **8**

Example

KB:

American(x) ∧Weapon(y) ∧ Sells(x, y, z) ∧ Hostile(z) ⇒ Criminal (x)	**1**
Owns(Nono,M1)	2
Missile(M1)	3
Missile(x) ∧ Owns(Nono, x) ⇒ Sells(West, x, Nono)	**4**
Missile(x) ⇒ Weapon(x)	**5**
Enemy(x,America) ⇒ Hostile(x)	**6**
American(West) .	7
Enemy(Nono, America)	8

A simple forward-chaining algorithm:

Starting from the known facts, it triggers all the rules whose premises are satisfied, adding their conclusions to the known facts.

The process repeats until the query is answered (assuming that just one answer is required) or no new facts are added.

Example – Forward Chaining

KB:

American(x) ∧ Weapon(y) ∧ Sells(x, y, z) ∧ Hostile(z) ⇒ Criminal (x)	**1**
Owns(Nono,M1)	2
Missile(M1)	3
Missile(x) ∧ Owns(Nono, x) ⇒ Sells(West, x, Nono)	**4**
Missile(x) ⇒ Weapon(x)	**5**
Enemy(x,America) ⇒ Hostile(x)	**6**
American(West) .	7
Enemy(Nono, America)	8

On the first iteration, rule 1 has unsatisfied premises.

Rule 4 is satisfied with {x/M1}, and Sells(West,M1, Nono) is added.

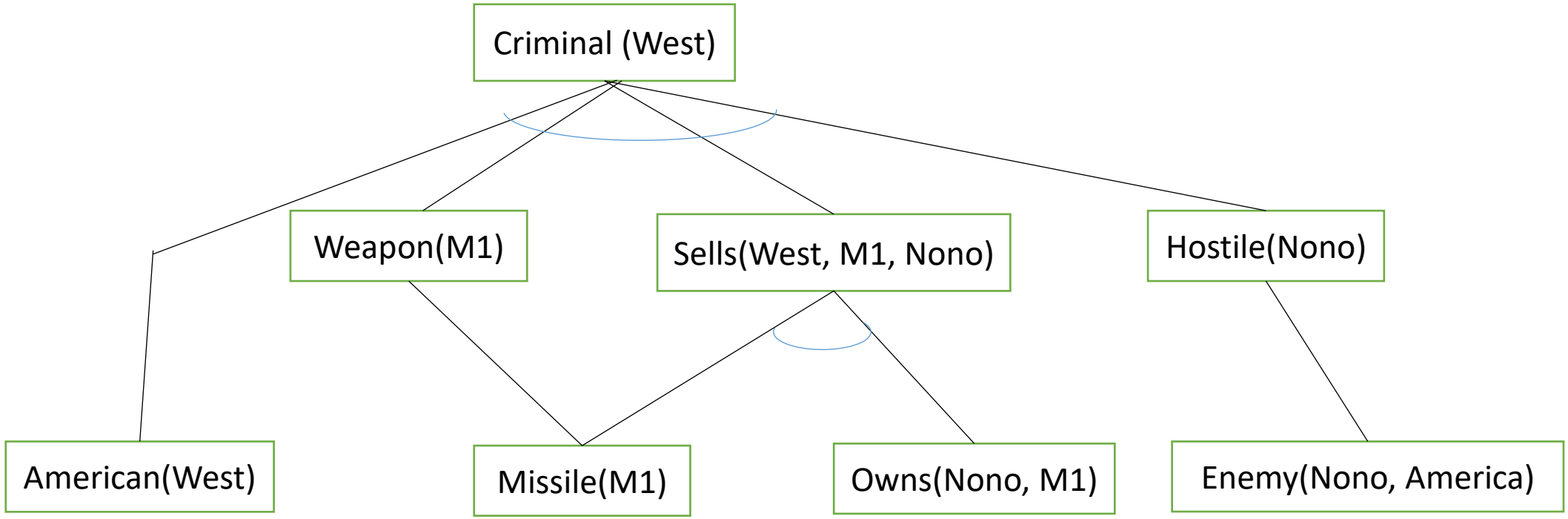
Rule 5 is satisfied with {x/M1}, and Weapon(M1) is added.

Rule 6 is satisfied with {x/Nono}, and Hostile(Nono) is added.

• On the second iteration, rule 1 is satisfied with {x/West, y/M1, z/Nono}, and Criminal (West) is added.

Example – Forward Chaining

Starting from the known facts, it triggers all the rules whose premises are satisfied, adding their conclusions to the known facts. The process repeats until the query is answered



The proof tree generated by forward chaining on the crime example.
The initial facts appear at the bottom level,
facts inferred on the first iteration in the middle level,
And facts inferred on the second iteration at the top level.

Example – Forward Chaining

Forward chaining is sound, because every inference is just an application of Generalized Modus Ponens, which is sound. Second,

it is complete for definite clause knowledge bases; that is, it answers every query whose answers are entailed by any knowledge base of definite clauses.

counting the number of possible facts that can be added, which determines the maximum number of iterations.

Let k be the maximum arity (number of arguments) of any predicate,

p be the number of predicates,

n be the number of constant symbols.

there can be no more than pn^k distinct ground facts,

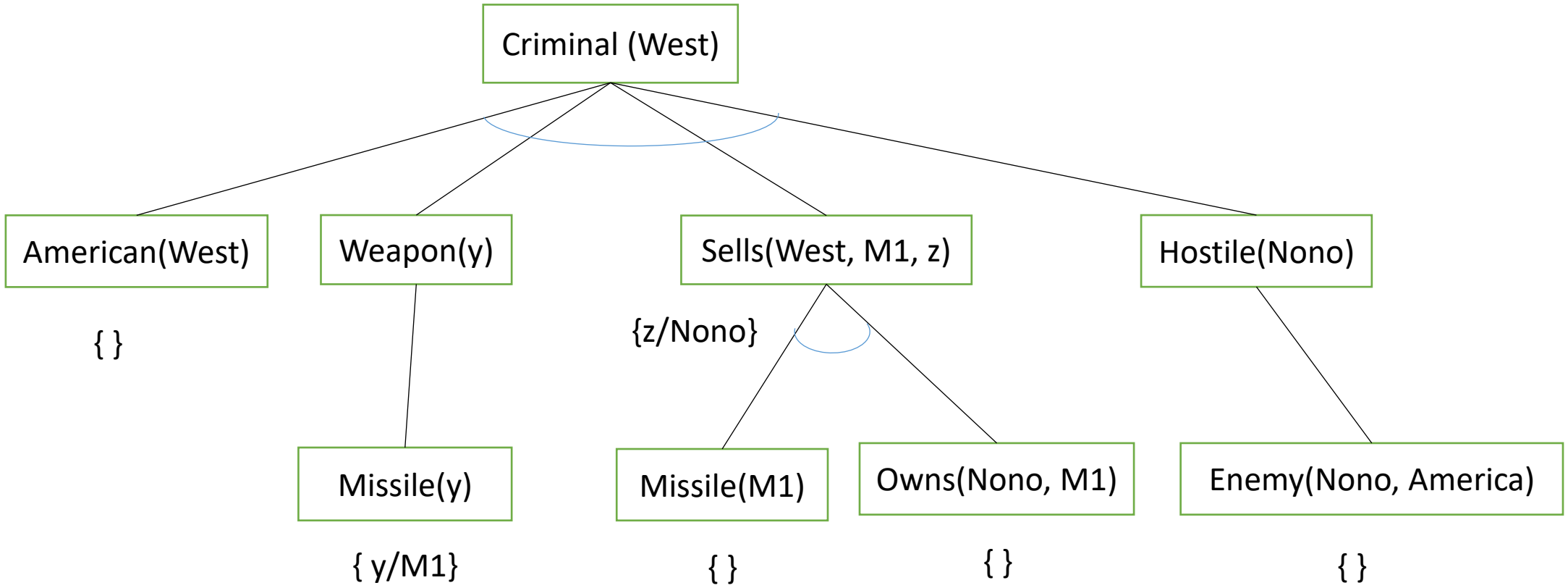
so after this many iterations the algorithm must have reached a fixed point.

no new inferences are possible at this point because every sentence that could be concluded by forward chaining is already contained explicitly in the KB. Such a knowledge base is called a fixed point of the inference process.

Repeated rule matches in each iteration, irrelevant facts are issues with simple Forward chaining. Iterative Forward Chaining, using partly goal information can reduce the complexity

Example – Backward Chaining

Algorithms work backward from the goal, chaining through rules to find known facts that support the proof.



Example – Backward Chaining

Proof tree constructed by backward chaining to prove that West is a criminal.

The tree should be read depth first, left to right.

To prove Criminal (West), we have to prove the four conjuncts below it.

Some of these are in the knowledge base, and others require further backward chaining.

Bindings for each successful unification are shown next to the corresponding subgoal.

Note that once one subgoal in a conjunction succeeds, its substitution is applied to subsequent subgoals.

Thus, by the time FOL-BC-ASK gets to the last conjunct, originally Hostile(z), z is already bound to Nono.

Backward Chaining – Algorithm

```
function FOL-BC-ASK(KB, query) returns a generator of substitutions
        return FOL-BC-OR(KB, query, { })
```

```
generator FOL-BC-OR(KB, goal , θ) yields a substitution
        for each rule (lhs ⇒ rhs) in FETCH-RULES-FOR-GOAL(KB, goal ) do
                (lhs, rhs)←STANDARDIZE-VARIABLES((lhs, rhs))
                for each θ′ in FOL-BC-AND(KB, lhs, UNIFY(rhs, goal , θ)) do
                yield θ′
```

```
generator FOL-BC-AND(KB, goals, θ) yields a substitution
        if θ = failure then return
        else if LENGTH(goals) = 0 then yield θ
        else do
        first,rest ←FIRST(goals), REST(goals)
        for each θ′ in FOL-BC-OR(KB, SUBST(θ, first), θ) do
                for each θ′′ in FOL-BC-AND(KB, rest , θ′) do
                        yield θ′′
```

Backward Chaining – Algorithm

Backward chaining is a kind of AND/OR search—

the OR part because the goal query can be proved by any rule in the knowledge base, and the AND part because all the conjuncts in the lhs of a clause must be proved.

FOL-BC-OR works by fetching all clauses that might unify with the goal,

standardizing the variables in the clause to be brand-new variables, and

then, if the rhs of the clause does indeed unify with the goal, proving every conjunct in the lhs, using FOL-BC-AND.

FOL-BC-AND in turn works by proving each of the conjuncts in turn, keeping track of the accumulated substitution as we go.

Backward chaining, works like a depth-first search algorithm.

This means that its space requirements are linear in the size of the proof

It also means that backward chaining (unlike forward chaining) suffers from problems with repeated states and incompleteness.

Resolution

As in the propositional case, first-order resolution requires that sentences be in conjunctive normal form (CNF)—that is, a conjunction of clauses, where each clause is a disjunction of literals.

Literals can contain variables, which are assumed to be universally quantified.

Example:

$\forall$ x American(x) $\wedge$Weapon(y) $\wedge$ Sells(x, y, z) $\wedge$ Hostile(z) $\Rightarrow$ Criminal (x)

becomes, in CNF,

$\neg$American(x) $\vee$ $\neg$Weapon(y) $\vee$ $\neg$Sells(x, y, z) $\vee$ $\neg$Hostile(z) $\vee$ Criminal (x) .

Every sentence of first-order logic can be converted into an inferentially equivalent CNF sentence.

CNF sentence will be unsatisfiable just when the original sentence is unsatisfiable,

so we have a basis for doing proofs by contradiction on the CNF sentences.

Conversion to CNF Process

1. Eliminate existential Quantifiers

Example:

$\forall x [\forall y \text{ Animal}(y) \Rightarrow \text{Loves}(x, y)] \Rightarrow [\exists y \text{ Loves}(y, x)]$.

• Eliminate implications:

$$\forall x [\neg \forall y \neg \text{Animal}(y) \vee \text{Loves}(x, y)] \vee [\exists y \text{ Loves}(y, x)] .$$

• Move ¬ inwards: In addition to the usual rules for negated connectives, we need rules for negated quantifiers. Thus, we have

$\neg \forall x\ p$ becomes $\exists x \neg p$

$\neg \exists x\ p$ becomes $\forall x \neg p$

sentence goes through the following transformations:

$$\forall x [\exists y \neg(\neg \text{Animal}(y) \vee \text{Loves}(x, y))] \vee [\exists y \text{ Loves}(y, x)] .$$

$$\forall x [\exists y \neg\neg \text{Animal}(y) \wedge \neg \text{Loves}(x, y)] \vee [\exists y \text{ Loves}(y, x)] .$$

$$\forall x [\exists y \text{ Animal } (y) \wedge \neg \text{Loves}(x, y)] \vee [\exists y \text{ Loves}(y, x)] .$$

universal quantifier ($\forall y$) in the premise of the implication has become an existential quantifier. The sentence now reads “Either there is some animal that x doesn’t love, or (if this is not the case) someone loves x.” Clearly, the meaning of the

original sentence has been preserved.

Conversion to CNF Process

- Standardize variables:

For sentences like

$$(\exists x\ P(x)) \vee (\exists x\ Q(x))$$

which use the same variable name twice, change the name of one of the variables, to avoid confusion later when the quantifiers are dropped.

$$\forall x\ [\exists y\ Animal(y) \wedge \neg Loves(x, y)] \vee [\exists z\ Loves(z, x)]$$

- Skolemize: Skolemization is the process of removing existential quantifiers by elimination.

  Existential Instantiation rule translate $\exists x\ P(x)$ into $P(A)$, where A is a new constant .

  given sentence doesn't match the pattern $\exists v\ \alpha$; if applied blindly,

  $\forall x\ [Animal(A) \wedge \neg Loves(x,A)] \vee Loves(B, x)$,

  everyone either fails to love a particular animal A or is loved by some particular entity B

original sentence allows each person to fail to love a different animal or to be loved by a different person.

Skolem entities need to depend on x and z:

$\forall x\ [Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee Loves(G(z), x)$. F and G are Skolem functions.

Conversion to CNF Process

The general rule is that the arguments of the Skolem function are all the universally quantified variables in whose scope the existential quantifier appears.

As with Existential Instantiation, the Skolemized sentence is satisfiable exactly when the original sentence is satisfiable.

- Drop universal quantifiers: At this point, all remaining variables must be universally quantified. Moreover, the sentence is equivalent to one in which all the universal quantifiers have been moved to the left. We can therefore drop the universal quantifiers:

  $$[\text{Animal}(F(x)) \land \neg \text{Loves}(x, F(x))] \lor \text{Loves}(G(z), x) .$$

- Distribute $\lor$ over $\land$:

  $$[\text{Animal}(F(x)) \lor \text{Loves}(G(z), x)] \land [\neg \text{Loves}(x, F(x)) \lor \text{Loves}(G(z), x)] .$$

Resolution Inference Rule

Resolution rule for first-order clauses is simply a lifted version of the propositional resolution rule

Two clauses, which are assumed to be standardized apart so that they share no variables, can be resolved if they contain complementary literals.

Propositional literals are complementary if one is the negation of the other;

first-order literals are complementary if one unifies with the negation of the other.

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

where $\text{UNIFY}(l_i, \neg m_j) = \theta$

This rule is called the binary resolution rule because it resolves exactly two literals.

The full resolution rule resolves subsets of literals in each clause that are unifiable.

Resolution Proof