Grundlagen der künstlichen Intelligenz – Inference in First-Order Logic

Matthias Althoff

TU München

November 21, 2019

KI_11/21

KI_11/28

Organization

1. Reducing First-Order Inference to Propositional Inference
2. Unification and Lifting
3. Forward Chaining
4. Backward Chaining
5. Resolution

The content is covered in the AI book by the section "Inference in First-Order Logic".

Learning Outcomes

Exam - related

- You can eliminate existential and universal quantifiers.
- You understand the problems of *propositionalization*.
- You can apply *Generalized Modus Ponens* and *Unification*.
- You can apply *forward chaining* and *backward chaining*.
- You understand the properties of *forward chaining* and *backward chaining*.
- You understand the basics of the syntax and semantics of *Prolog*.
- You can transform any first-order logic sentence into *Conjunctive Normal Form*.
- You can apply *resolution* in first-order logic.

Removing Quantifiers in First-Order Logic

- We present simple inference rules that can remove quantifiers to obtain sentences without quantifiers.
- This naturally leads to converting first-order inference to propositional inference, which we already know.
- We begin with removing universal quantifiers, then existential quantifiers, and finally discuss the result.

Universal Instantiation (UI)

1

Let's start with the axiom that all greedy kings are evil:

$$\forall\, x \quad King(x) \wedge Greedy(x) \Rightarrow Evil(x).$$

It seems quite permissible to infer any of the following sentences:

$$King(John) \wedge Greedy(John) \Rightarrow Evil(John)$$
$$King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$$
$$King(Father(John)) \wedge Greedy(Father(John)) \Rightarrow Evil(Father(John))$$
$$\vdots$$

- The rule of **universal instantiation** says that we can infer any sentence from substituting a **ground term** (a term without variables) for the variable.
- Let $\texttt{Subst}(\theta, \alpha)$ denote the result of applying the substitution θ to the sentence α; we write

$$\frac{\forall\, v \quad \alpha}{\texttt{Subst}(\{v/g\}, \alpha)}.$$

- The above sentences are obtained from the substitutions $\{x/John\}$, $\{x/Richard\}$, and $\{x/Father(John)\}$.

Existential Instantiation ② (EI)

- When existential quantifiers appear, we replace a variable by a single *new constant symbol*.
- More formally: For any sentence α, variable v, and constant symbol k *that does not appear elsewhere in the knowledge base*:

$$\frac{\exists\, v \quad \alpha}{\texttt{Subst}(\{v/k\}, \alpha)}$$

- E.g., $\exists\, x \quad Crown(x) \land OnHead(x, John)$ yields

$$Crown(C_1) \land OnHead(C_1, John)$$

not used before in KB

provided C_1 is a new constant symbol, called a **Skolem constant**.

- Another example: from $\exists\, x \quad d(x^y)/dy = x^y$ we obtain

$$d(e^y)/dy = e^y$$

provided e (Euler's number) is a new constant symbol, which differs from e.g., π.

Existential Instantiation: Change of Knowledge Base

Important

Universal Instantiation

- Universal instantiation can be applied several times to **add** sentences.
- The new knowledge base is logically equivalent to the old when
  - performing **all** possible substitutions and deleting the quantified sentence, or
  - adding new sentences and keeping the quantified sentence.

Existential Instantiation

- Existential instantiation can be applied once to **replace** the existential sentence.
- The new knowledge base is **not** equivalent to the old, but is satisfiable iff the old knowledge base was satisfiable.

Reduction to Propositional Inference: Example

Suppose the knowledge base contains just the following:

$$\forall x \quad King(x) \wedge Greedy(x) \Rightarrow Evil(x)$$
$$King(John)$$
$$Greedy(John)$$
$$Brother(Richard, John)$$

Instantiating the universal sentence in **all possible** ways, we have

we removed $\forall x$

$$King(John) \wedge Greedy(John) \Rightarrow Evil(John)$$
$$King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$$
$$King(John)$$
$$Greedy(John)$$
$$Brother(Richard, John)$$

The new knowledge base is **propositionalized**: proposition symbols are

$King(John)$, $Greedy(John)$, $Evil(John)$, $King(Richard)$, etc.

Reduction to Propositional Inference: General Approach

Key

- **Claim**: Every FOL KB can be propositionalized so as to preserve entailment.
- **Idea**: Propositionalize KB and query, apply resolution, return result.
- **Problem**: With function symbols, there are infinitely many ground terms, e.g., *Father*(*Father*(*Father*(*John*))).
- **Theorem**: Herbrand (1930). If a sentence α is entailed by a FOL KB, it is entailed by a **finite** subset of the propositional KB.
- **Idea**: For $n = 0$ to ∞ do:
  Create a propositional KB by instantiating with depth-n terms and see if α is entailed by this KB. Simply, you will not be sure whether the statement itself is incorrect or simply because you didn't try hard enough
- **Problem**: Works if α is entailed, loops if α is not entailed.
- **Theorem**: Turing (1936), Church (1936), entailment in FOL is **semidecidable** (algorithms exist that say yes to every entailed sentence, but no algorithm exists that also says no to every nonentailed sentence).

Problems with Propositionalization

- Propositionalization seems to generate lots of irrelevant sentences.
  E.g., from

  $$\begin{aligned}&\forall x \quad King(x) \wedge Greedy(x) \Rightarrow Evil(x)\\&King(John)\\&\forall y \quad Greedy(y)\\&Brother(Richard, John)\end{aligned}$$

  it seems obvious that $Evil(John)$, but propositionalization produces lots of facts such as $Greedy(Richard)$ that are irrelevant.
- With p k-ary predicates and n constants, there are $p\,n^k$ instantiations.
- With function symbols, it gets much worse!

A First-Order Inference Rule

- Due to the presented problems with propositionalization, we aim at directly inferring sentences in first-order logic.
- To infer $Evil(John)$ from a simplified version of the previous example

$$\begin{aligned}&\forall x \quad King(x) \wedge Greedy(x) \Rightarrow Evil(x)\\&King(John)\\&Greedy(John)\end{aligned}$$

  we only need to find a substitution θ that makes $King(x)$, $Greedy(x)$ identical to sentences already in the KB, so that we can assert the conclusion $Evil(x)$.
  Solution: $\theta = \{x/John\}$.
- This inference process is called **Generalized Modus Ponens**.

Generalized Modus Ponens (GMP)

Important

For atomic sentences p_i, p_i', and q, where there is for all i a substitution θ such that $\mathtt{Subst}(\theta, p_i') = \mathtt{Subst}(\theta, p_i)$, we have that

*

assuming that universal quantifier is included

$$\frac{p_1{}',\ \ p_2{}',\ \ldots,\ p_n{}',\ \ (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{\mathtt{Subst}(\theta, q)}$$

Example:

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

$$King(John)$$

$\forall\, y \quad Greedy(y)$ (this line changed from previous example)

Show that Evil(John) with GMP:

$p_1{}'$ is $King(John)$ $\qquad$ p_1 is $King(x)$

$p_2{}'$ is $Greedy(y)$ $\qquad$ p_2 is $Greedy(x)$

q is $Evil(x)$

θ is $\{x/John, y/John\}$

$\mathtt{Subst}(\theta, q)$ is $Evil(John)$

Soundness of GMP

We need to show that

$$p_1', \ldots, p_n', \ (p_1 \land \ldots \land p_n \Rightarrow q) \models \texttt{Subst}(\theta, q)$$

provided that $\texttt{Subst}(\theta, p_i') = \texttt{Subst}(\theta, p_i)$ for all i.

Lemma

For any sentence p (whose variables are assumed to be universally quantified), we have $p \models \texttt{Subst}(\theta, p)$ by universal instantiation.

1. $(p_1 \land \ldots \land p_n \Rightarrow q) \models (\texttt{Subst}(\theta, p_1) \land \ldots \land \texttt{Subst}(\theta, p_n) \Rightarrow \texttt{Subst}(\theta, q))$
2. $p_1', \ldots, p_n' \models p_1' \land \ldots \land p_n' \models \texttt{Subst}(\theta, p_1') \land \ldots \land \texttt{Subst}(\theta, p_n')$
3. From $\texttt{Subst}(\theta, p_i') = \texttt{Subst}(\theta, p_i)$ and steps 1, 2, ordinary Modus Ponens results in $\texttt{Subst}(\theta, q)$.

Generalized Modus Ponens is a **lifted** version of Modus Ponens – it raises Modus Ponens from propositional logic to first-order logic.

Unification

- Lifted inference rules require finding substitutions that make different logical expressions look identical, called **unification**.
- The unify algorithm $\texttt{Unify}(p, q) = \theta$ returns a unifier θ such that $\texttt{Subst}(\theta, p) = \texttt{Subst}(\theta, q)$ if it exists.

Example:

Variable

Specific constant

p	**q**	$\boldsymbol{\theta}$
$Knows(John, x)$	$Knows(John, Jane)$	$\{x/Jane\}$
$Knows(John, x)$	$Knows(y, Elizabeth)$	$\{x/Elizabeth, y/John\}$
$Knows(John, x)$	$Knows(y, Mother(y))$	$\{y/John, x/Mother(John)\}$
$Knows(John, x)$	$Knows(x, Elizabeth)$	*fail*

- The last unification fails because x cannot take the values *John* and *Elizabeth* at the same time.
- The problem can be avoided by **standardizing apart** one of the two sentences by renaming its variables, e.g., $Knows(x_{17}, Elizabeth)$ instead of $Knows(x, Elizabeth)$ in the last line.

Tweedback Questions

* A Horn clause is a clause (a disjunction of literals) with at most one positive, i.e unnegated, literal

– Example: Definite clause "exactly one +ve literal"

- **Unify**
  $parents(x, father(x), mother(Bill))$ and $parents(Bill, father(Bill), y)$:
  - A $\{x/Bill, y/mother(z)\}$
  - B $\{x/Bill, y/mother(Bill)\}$
- **Unify**
  $parents(x, father(x), mother(Bill))$ and $parents(Bill, father(y), z)$:
  - A $x/Bill, y/Bill, z/mother(Bill)$
  - B $x/Bill, x/y, z/mother(Bill)$

disjunction form: $\neg p \vee \neg q \vee \dots \vee \neg t \vee u$

implication form: $u \leftarrow p \wedge q \wedge \dots \wedge t$

read intuitively as: assume that, if p & q & ... & t all hold, then also u holds

Unification: Most General Unifier

- In many cases, there is more than one unifier, e.g.,

$$\texttt{Unify}(Knows(John, x), Knows(y, z))$$

could return

$$\{y/John, x/z\} \text{ or } \{y/John, x/John, z/John\}.$$

- The first unifier gives $Knows(John, z)$ as the result, the second one gives $Knows(John, John)$. The second result can be obtained from the first one by the substitution $z/John$
  → The first unifier is more general. [there are many possibilities]
- There always exists a **most general unifier** as shown by the algorithm in Fig. 1 of the AI book in the section "Inference in First-Order Logic".
- If variables are replaced by variables, we replace the ones of the first sentence with the ones of the second sentence.

First-Order Horn Clauses

As for propositional logic, Horn clauses allow one to use forward and backward chaining – a very efficient inference technique.
Reminder:

Horn clause in propositional logic

- proposition symbol; or
- (conjunction of symbols) $\Rightarrow$ symbol

The difference in first-order logic is simply that universally quantified variables are allowed (the universal quantifier is typically omitted when writing Horn clauses). yet always assumed

Example:

$$\forall x \quad King(x) \wedge Greedy(x) \Rightarrow Evil(x)$$
$$King(John)$$
$$\forall y \quad Greedy(y)$$

Forward-Chaining: Main Idea

1st inference method

- Starting from the known facts, forward chaining triggers all the rules whose premises are satisfied and adds their conclusions to the known facts.
- The process repeats until the query is answered or no new facts are added.
- A fact is not new if it is just a **renaming** of an old fact, e.g., $Likes(x, IceCream)$ and $Likes(y, IceCream)$ have identical meaning.
- We introduce $\texttt{Standardize} - \texttt{Apart}(r)$ of a sentence r, which renames all variables with variables that have not been used before to avoid unification issues, such as $\texttt{Unify}(Knows(John, x), Knows(x, Elizabeth)) = fail$.

Forward-Chaining Algorithm (ForwardChaining.ipynb)

function FOL-FC-Ask (KB,α) **returns** a substitution or *false*

Statement to prove true or false

repeat until *new* is empty

new $\leftarrow \emptyset$

for each sentence r **in** KB **do**

$(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow \texttt{Standardize-Apart}(r)$

GMP

for each θ such that $\texttt{Subst}(\theta, p_1 \wedge \ldots \wedge p_n) = \texttt{Subst}(\theta, p_1' \wedge \ldots \wedge p_n')$

for some $p_1', \ldots, p_n'$ in KB

$q' \leftarrow \texttt{Subst}(\theta,\ q)$

if q' is not a renaming of a sentence already in KB or *new* **then do**

add q' to *new*

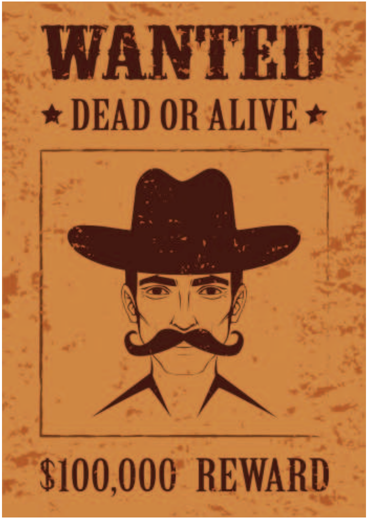
$\phi \leftarrow \texttt{Unify}(q',\alpha)$

if ϕ is not *fail* **then return** ϕ

add *new* to KB

return *false*

Example: Criminal West



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 Col. West is criminal.

First-Order Horn Clauses: Criminal West

- ... **it is a crime for an American to sell weapons to hostile nations**:
'$\forall x$' $American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$
- **Nono ... has some missiles, i.e.,** $\exists\, x\ Owns(Nono, x) \wedge Missile(x)$: x is already used
$Owns(Nono, M_1)$ and $Missile(M_1)$ (using existential instantiation)
- ... **all of its missiles were sold to it by Colonel West**:
$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
- **Missiles are weapons**:
$Missile(x) \Rightarrow Weapon(x)$
- **An enemy of America counts as "hostile"**:
$Enemy(x, America) \Rightarrow Hostile(x)$
- **West, who is American** ...
$American(West)$
- **The country Nono, an enemy of America** ...
$Enemy(Nono, America)$

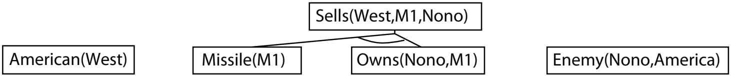
Q: which premise to start with?

A: mainly intuition

Criminal West: Forward Chaining (1a)

function FOL-FC-Ask (KB,α) **returns** a substitution or *false*

repeat until *new* is empty
    new $\leftarrow \emptyset$
    for each sentence r **in** KB **do**
        $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow \texttt{Standardize-Apart}(r)$
        for each θ such that $\texttt{Subst}(\theta, p_1 \wedge .. \wedge p_n) = \texttt{Subst}(\theta, p'_1 \wedge .. \wedge p'_n)$ for some $p'_1, .., p'_n$ in KB
            $q' \leftarrow \texttt{Subst}(\theta,\ q)$
            if q' is not a renaming of a sentence already in KB or *new* **then do**
                add q' to *new*
                $\phi \leftarrow \texttt{Unify}(q',\alpha)$
                if ϕ is not *fail* **then return** ϕ
    add *new* to KB
return *false*

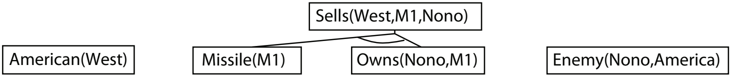


Used clause (after Standardize-Apart):
$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

Criminal West: Forward Chaining (1b)

function FOL-FC-Ask (KB,α) **returns** a substitution or *false*

repeat until *new* is empty
    new $\leftarrow \emptyset$
    for each sentence r **in** KB **do**
        $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ `Standardize-Apart`(r)
        for each θ such that `Subst`$(\theta, p_1 \wedge .. \wedge p_n)$ = `Subst`$(\theta, p'_1 \wedge .. \wedge p'_n)$ for some $p'_1, .., p'_n$ in KB
            $q' \leftarrow$ `Subst`$(\theta,\ q)$
            if q' is not a renaming of a sentence already in KB or *new* **then do**
                add q' to *new*
                $\phi \leftarrow$ `Unify`(q',α)
                if ϕ is not *fail* **then return** ϕ
    add *new* to KB
return *false*

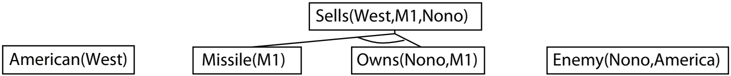


Used clause (after `Standardize-Apart`):
$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$ We are trying to align Knowledge with premise
The premise is satisfied with $\theta = \{x/M_1\}$

Criminal West: Forward Chaining (1c)

function FOL-FC-Ask (KB,α) **returns** a substitution or *false*

repeat until *new* is empty
    new $\leftarrow \emptyset$
    for each sentence r **in** KB **do**
        $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow \texttt{Standardize-Apart}(r)$
        for each θ such that $\texttt{Subst}(\theta, p_1 \wedge .. \wedge p_n) = \texttt{Subst}(\theta, p_1' \wedge .. \wedge p_n')$ for some $p_1', .., p_n'$ in KB
            $q' \leftarrow \texttt{Subst}(\theta,\ q)$
            if q' is not a renaming of a sentence already in KB or *new* **then do**
                add q' to *new*
                $\phi \leftarrow \texttt{Unify}(q',\alpha)$
                if ϕ is not *fail* **then return** ϕ
    add *new* to KB
return *false*



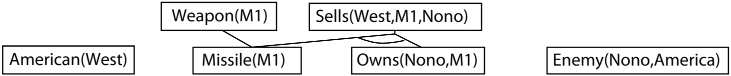
Used clause (after `Standardize-Apart`):
$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$ → will become : Sells (West, M1, Nono)

The premise is satisfied with $\theta = \{x/M_1\}$
$q' = Sells(West, M_1, Nono)$ is added.   q` is added to the knowledge base

Criminal West: Forward Chaining (2a)

function FOL-FC-Ask (KB,α) **returns** a substitution or *false*

repeat until *new* is empty
    $new \leftarrow \emptyset$
    for each sentence r **in** KB **do**
        $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow \texttt{Standardize-Apart}(r)$
        for each θ such that $\texttt{Subst}(\theta, p_1 \wedge .. \wedge p_n) = \texttt{Subst}(\theta, p'_1 \wedge .. \wedge p'_n)$ for some $p'_1, .., p'_n$ in KB
            $q' \leftarrow \texttt{Subst}(\theta,\ q)$
            if q' is not a renaming of a sentence already in KB or *new* **then do**
                add q' to *new*
                $\phi \leftarrow \texttt{Unify}(q',\alpha)$
                if ϕ is not *fail* **then return** ϕ
    add *new* to KB
return *false*

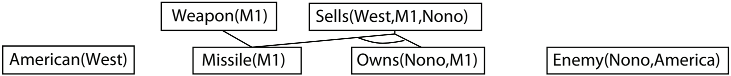


Used clause (after Standardize-Apart):
$Missile(x) \Rightarrow Weapon(x)$

Criminal West: Forward Chaining (2b)

function FOL-FC-Ask (KB,α) **returns** a substitution or *false*

repeat until *new* is empty
    $new \leftarrow \emptyset$
    for each sentence r **in** KB **do**
        $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow \texttt{Standardize-Apart}(r)$
        for each θ such that $\texttt{Subst}(\theta, p_1 \wedge .. \wedge p_n) = \texttt{Subst}(\theta, p'_1 \wedge .. \wedge p'_n)$ for some $p'_1, .., p'_n$ in KB
            $q' \leftarrow \texttt{Subst}(\theta,\ q)$
            if q' is not a renaming of a sentence already in KB or *new* **then do**
                add q' to *new*
                $\phi \leftarrow \texttt{Unify}(q',\alpha)$
                if ϕ is not *fail* **then return** ϕ
    add *new* to KB
return *false*



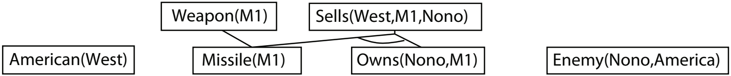
Used clause (after Standardize-Apart):
$Missile(x) \Rightarrow Weapon(x)$

The premise is satisfied with $\theta = \{x/M_1\}$

Criminal West: Forward Chaining (2c)

function FOL-FC-Ask (KB,α) **returns** a substitution or *false*

repeat until *new* is empty
  $new \leftarrow \emptyset$
  for each sentence r **in** KB **do**
    $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ `Standardize-Apart`(r)
    for each θ such that `Subst`$(\theta, p_1 \wedge .. \wedge p_n) =$ `Subst`$(\theta, p'_1 \wedge .. \wedge p'_n)$ for some $p'_1, .., p'_n$ in KB
      $q' \leftarrow$ `Subst`$(\theta,\ q)$
      if q' is not a renaming of a sentence already in KB or *new* **then do**
        add q' to *new*
        $\phi \leftarrow$ `Unify`(q',α)
        if ϕ is not *fail* **then return** ϕ
  add *new* to KB
return *false*



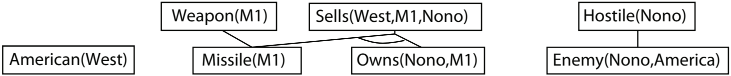
Used clause (after `Standardize-Apart`):
$Missile(x) \Rightarrow Weapon(x)$

The premise is satisfied with $\theta = \{x/M_1\}$
$q' = Weapon(M_1)$ is added.

Criminal West: Forward Chaining (3)

function FOL-FC-Ask (KB, α) **returns** a substitution or *false*

repeat until *new* is empty
  new $\leftarrow \emptyset$
  for each sentence r **in** KB **do**
    $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ `Standardize-Apart`(r)
    for each θ such that `Subst`$(\theta, p_1 \wedge .. \wedge p_n)$ = `Subst`$(\theta, p'_1 \wedge .. \wedge p'_n)$ for some $p'_1, .., p'_n$ in KB
      $q' \leftarrow$ `Subst`$(\theta,\ q)$
      if q' is not a renaming of a sentence already in KB or *new* **then do**
        add q' to *new*
        $\phi \leftarrow$ `Unify`(q', α)
        if ϕ is not *fail* **then return** ϕ
  add *new* to KB
return *false*



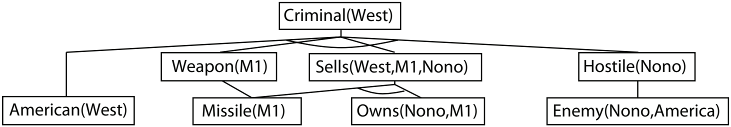
Used clause (after `Standardize-Apart`):
$Enemy(x, America) \Rightarrow Hostile(x)$

The premise is satisfied with $\theta = \{x/Nono\}$
$q' = Hostile(Nono)$ is added.

Criminal West: Forward Chaining (4)

function FOL-FC-Ask (*KB*,α) **returns** a substitution or *false*

```
repeat until new is empty
    new ← ∅
    for each sentence r in KB do
        (p1 ∧ ... ∧ pn ⇒ q) ← Standardize-Apart(r)
        for each θ such that Subst(θ, p1 ∧ .. ∧ pn) = Subst(θ, p'1 ∧ .. ∧ p'n) for some p'1, .., p'n in KB
            q' ← Subst(θ, q)
            if q' is not a renaming of a sentence already in KB or new then do
                add q' to new
                φ ← Unify(q',α)
                if φ is not fail then return φ
    add new to KB
return false
```



Used clause (after `Standardize-Apart`):
$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

The premise is satisfied with $\theta = \{x/West, y/M_1, z/Nono\}$
$q' = Criminal(West)$ is added. $\rightarrow$ West is a criminal!

Properties of Forward Chaining

- Sound and complete for first-order definite clauses (proof similar to the one for propositional logic)
- Forward chaining may not terminate in general if function symbols are involved; e.g., Peano axiom for natural numbers:

$$NatNum(0)$$

$$\forall\, n \quad NatNum(n) \Rightarrow NatNum(S(n)) \qquad (S(n):\ \text{successor of } n)$$

Forward chaining would add $NatNum(S(0))$, $NatNum(S(S(0)))$, ...

- This is unavoidable: entailment with definite clauses is semidecidable (as for general first-order logic). 'running forever'

Datalog: Declarative logic programming language

Datalog = first-order definite clauses + *no functions* (e.g., crime KB)
Forward chaining terminates for Datalog in a polynomial number of iterations: at most $p \cdot n^k$ literals (p: number of predicates, k: maximum arity of predicates, n: number of constant symbols)

Backward-Chaining: Main Idea

2nd inference method

expansion is only in the necessary direction

- Starting from the goal, backward chaining searches rules to find known facts that support the proof.
- The process repeats until the query is answered or no new sub-goals can be added.
- Backwards chaining is the workhorse for logic programming.

Backward-Chaining Algorithm (BackwardChaining.ipynb)

function FOL-BC-Ask (KB,$goals$,θ) **returns** a set of substitutions

inputs: KB, a knowledge base
$goals$, a list of conjuncts forming a query (θ already applied)
θ, the current substitution, initially the empty substitution $\varnothing$

local variables: $answers$, a set of substitutions, initially empty

if $goals$ is empty **then return** $\{\theta\}$
$q' \leftarrow \texttt{Subst}(\theta, \texttt{First}(goals))$
for each sentence r **in** KB where $\texttt{Standardize-Apart}(r) = (p_1 \land \ldots \land p_n \Rightarrow q)$
and $\theta' \leftarrow \texttt{Unify}(q, q')$ succeeds
    $new\_goals \leftarrow [p_1, \ldots, p_n | \texttt{Rest}(goals)]$
    $answers \leftarrow \texttt{FOL-BC-Ask}(KB, new\_goals, \texttt{Compose}(\theta', \theta)) \cup answers$
return $answers$

Criminal West: Backward Chaining (1a)

function FOL-BC-Ask (KB,$goals$,θ) **returns** a set of substitutions

if $goals$ is empty **then return** $\{\theta\}$
$q' \leftarrow \texttt{Subst}(\theta, \texttt{First}(goals))$
for each sentence r **in** KB where $\texttt{Standardize-Apart}(r) = (p_1 \wedge \ldots \wedge p_n \Rightarrow q)$ and $\theta' \leftarrow \texttt{Unify}(q, q')$ succeeds
    $new\_goals \leftarrow [p_1, \ldots, p_n | \texttt{Rest}(goals)]$
    $answers \leftarrow \texttt{FOL-BC-Ask}(KB, new\_goals, \texttt{Compose}(\theta', \theta)) \cup answers$
return $answers$

$\boxed{\text{Criminal(West)}}$

$goals$: $\{Criminal(West)\}$
$q' \leftarrow \texttt{Subst}(\varnothing, \text{Criminal(West)})$

Criminal West: Backward Chaining (1b)

function FOL-BC-Ask (KB,$goals$,θ) **returns** a set of substitutions

if *goals* is empty **then return** $\{\theta\}$
$q' \leftarrow \texttt{Subst}(\theta, \texttt{First}(goals))$
for each sentence r **in** KB where $\texttt{Standardize-Apart}(r) = (p_1 \wedge \ldots \wedge p_n \Rightarrow q)$ and $\theta' \leftarrow \texttt{Unify}(q, q')$ succeeds
 $new\_goals \leftarrow [p_1, \ldots, p_n | \texttt{Rest}(goals)]$
 $answers \leftarrow \texttt{FOL-BC-Ask}(KB, new\_goals, \texttt{Compose}(\theta', \theta)) \cup answers$
return *answers*

to avoid naming different variables with the same name

$\boxed{\text{Criminal(West)}}$

$goals$: $\{Criminal(West)\}$
$q' \leftarrow \texttt{Subst}(\varnothing, \text{Criminal(West)})$
$\theta' \leftarrow \{x_1/West\}$

then add all premises to new goals

Used clause (after Standardize-Apart):
$American(x_1) \wedge Weapon(y_1) \wedge Sells(x_1, y_1, z_1) \wedge Hostile(z_1) \Rightarrow Criminal(x_1)$

Criminal West: Backward Chaining (1c)

function FOL-BC-Ask (KB,$goals$,θ) **returns** a set of substitutions

if *goals* is empty **then return** $\{\theta\}$
$q' \leftarrow \texttt{Subst}(\theta, \texttt{First}(goals))$
for each sentence r **in** KB where $\texttt{Standardize-Apart}(r) = (p_1 \wedge \ldots \wedge p_n \Rightarrow q)$ and $\theta' \leftarrow \texttt{Unify}(q, q')$ succeeds
  $new\_goals \leftarrow [p_1, \ldots, p_n | \texttt{Rest}(goals)]$
  $answers \leftarrow \texttt{FOL-BC-Ask}(KB, new\_goals, \texttt{Compose}(\theta', \theta)) \cup answers$
return *answers*

$\boxed{\text{Criminal(West)}}$

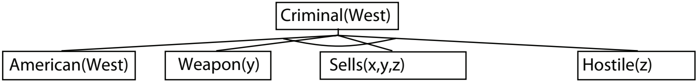
goals: $\{Criminal(West)\}$
$q' \leftarrow \texttt{Subst}(\varnothing, \text{Criminal(West)})$
$\theta' \leftarrow \{x_1/West\}$
$new\_goals \leftarrow \{American(x_1), Weapon(y_1), Sells(x_1, y_1, z_1), Hostile(z_1)\}$

Used clause (after Standardize-Apart):
$American(x_1) \wedge Weapon(y_1) \wedge Sells(x_1, y_1, z_1) \wedge Hostile(z_1) \Rightarrow Criminal(x_1)$

Criminal West: Backward Chaining (2a)

function FOL-BC-Ask (KB,$goals$,θ) **returns** a set of substitutions

if $goals$ is empty **then return** $\{\theta\}$
$q' \leftarrow \texttt{Subst}(\theta, \texttt{First}(goals))$
for each sentence r **in** KB where $\texttt{Standardize-Apart}(r) = (p_1 \wedge \ldots \wedge p_n \Rightarrow q)$ and $\theta' \leftarrow \texttt{Unify}(q, q')$ succeeds
    $new\_goals \leftarrow [p_1, \ldots, p_n | \texttt{Rest}(goals)]$
    $answers \leftarrow \texttt{FOL-BC-Ask}(KB, new\_goals, \texttt{Compose}(\theta', \theta)) \cup answers$
return $answers$



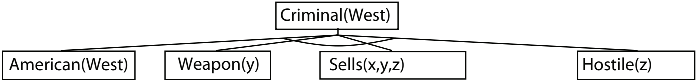
$goals$: $\{American(x_1), Weapon(y_1), Sells(x_1, y_1, z_1), Hostile(z_1)\}$
$q' \leftarrow \texttt{Subst}(\{x_1/West\}, American(x_1))$

we apply the substitution to the goals
one by one

Criminal West: Backward Chaining (2b)

function FOL-BC-Ask (KB,$goals$,θ) **returns** a set of substitutions

if *goals* is empty **then return** $\{\theta\}$
$q' \leftarrow$ `Subst`$(\theta,$ `First`$(goals))$
for each sentence r **in** KB where `Standardize-Apart`$(r) = (p_1 \land \ldots \land p_n \Rightarrow q)$ and $\theta' \leftarrow$ `Unify`(q, q') succeeds
  $new\_goals \leftarrow [p_1, \ldots, p_n|$`Rest`$(goals)]$
  $answers \leftarrow$ `FOL-BC-Ask`$(KB,new\_goals,$`Compose`$(\theta',\theta)) \cup answers$
return *answers*



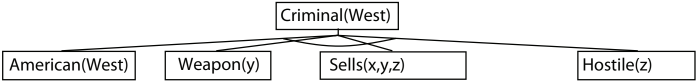
$goals$: $\{American(x_1), Weapon(y_1), Sells(x_1, y_1, z_1), Hostile(z_1)\}$
$q' \leftarrow$ `Subst`$(\{x_1/West\},\ American(x_1))$
$\theta' \leftarrow \emptyset$ nothing will ~~be~~ updated for θ'

Used clause (after `Standardize-Apart`):
$American(West)$ already in KB

Criminal West: Backward Chaining (2c)

function FOL-BC-Ask (KB,$goals$,θ) **returns** a set of substitutions

if *goals* is empty **then return** $\{\theta\}$
$q' \leftarrow \texttt{Subst}(\theta, \texttt{First}(goals))$
for each sentence r **in** KB where $\texttt{Standardize-Apart}(r) = (p_1 \wedge \ldots \wedge p_n \Rightarrow q)$ and $\theta' \leftarrow \texttt{Unify}(q, q')$ succeeds
    $new\_goals \leftarrow [p_1, \ldots, p_n | \texttt{Rest}(goals)]$
    $answers \leftarrow \texttt{FOL-BC-Ask}(KB, new\_goals, \texttt{Compose}(\theta', \theta)) \cup answers$
return *answers*



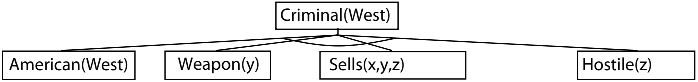
$goals$: $\{American(x_1), Weapon(y_1), Sells(x_1, y_1, z_1), Hostile(z_1)\}$
$q' \leftarrow \texttt{Subst}(\{x_1/West\}, American(x_1))$
$\theta' \leftarrow \emptyset$
$new\_goals \leftarrow \{Weapon(y_1), Sells(x_1, y_1, z_1), Hostile(z_1)\}$

Used clause (after Standardize-Apart):
$American(West)$

Criminal West: Backward Chaining (3)

function FOL-BC-Ask (KB,$goals$,θ) **returns** a set of substitutions

if *goals* is empty **then return** $\{\theta\}$
$q' \leftarrow$ `Subst`$(\theta,$ `First`$(goals))$
for each sentence r **in** KB where `Standardize-Apart`$(r) = (p_1 \wedge \ldots \wedge p_n \Rightarrow q)$ and $\theta' \leftarrow$ `Unify`(q, q') succeeds
    $new\_goals \leftarrow [p_1, \ldots, p_n|$`Rest`$(goals)]$
    $answers \leftarrow$ `FOL-BC-Ask`$(KB, new\_goals,$`Compose`$(\theta', \theta)) \cup answers$
return *answers*



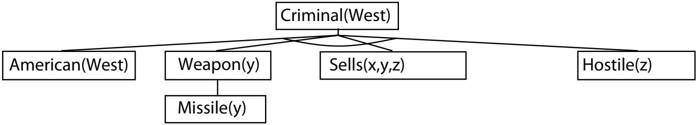
$goals$: $\{Weapon(y_1), Sells(x_1, y_1, z_1), Hostile(z_1)\}$
$q' \leftarrow$ `Subst`$(\{x_1/West\},\ Weapon(y_1))$
$\theta' \leftarrow \{x_2/y_1\}$
$new\_goals \leftarrow \{Missile(x_2), Sells(x_1, y_1, z_1), Hostile(z_1)\}$

Used clause (after `Standardize-Apart`):
$Missile(x_2) \Rightarrow Weapon(x_2)$

Criminal West: Backward Chaining (4)

function FOL-BC-Ask $(KB, goals, \theta)$ **returns** a set of substitutions

if *goals* is empty **then return** $\{\theta\}$
$q' \leftarrow \texttt{Subst}(\theta, \texttt{First}(goals))$
for each sentence r **in** KB where $\texttt{Standardize-Apart}(r) = (p_1 \wedge \ldots \wedge p_n \Rightarrow q)$ and $\theta' \leftarrow \texttt{Unify}(q, q')$ succeeds
  $new\_goals \leftarrow [p_1, \ldots, p_n | \texttt{Rest}(goals)]$
  $answers \leftarrow \texttt{FOL-BC-Ask}(KB, new\_goals, \texttt{Compose}(\theta', \theta)) \cup answers$
return *answers*



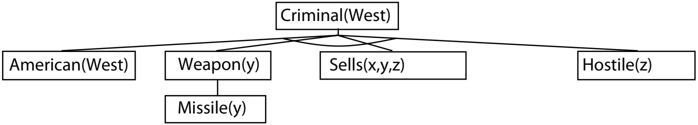
goals: $\{Missile(x_2), Sells(x_1, y_1, z_1), Hostile(z_1)\}$
$q' \leftarrow \texttt{Subst}(\{x_1/West, x_2/y_1\}, Missile(x_2))$
$\theta' \leftarrow \{y_1/M_1\}$
$new\_goals \leftarrow \{Sells(x_1, y_1, z_1), Hostile(z_1)\}$

Used clause (after Standardize-Apart):
$Missile(M_1)$

Criminal West: Backward Chaining (5)

function FOL-BC-Ask (KB,$goals$,θ) **returns** a set of substitutions

if *goals* is empty **then return** $\{\theta\}$
$q' \leftarrow \texttt{Subst}(\theta, \texttt{First}(goals))$
for each sentence r **in** KB where $\texttt{Standardize-Apart}(r) = (p_1 \wedge \ldots \wedge p_n \Rightarrow q)$ and $\theta' \leftarrow \texttt{Unify}(q, q')$ succeeds
    $new\_goals \leftarrow [p_1, \ldots, p_n|\texttt{Rest}(goals)]$
    $answers \leftarrow \texttt{FOL-BC-Ask}(KB, new\_goals, \texttt{Compose}(\theta', \theta)) \cup answers$
return *answers*



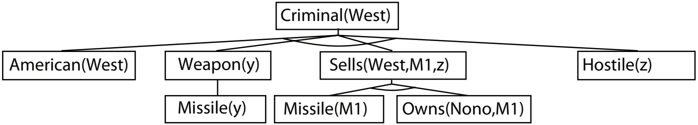
goals: $\{Sells(x_1, y_1, z_1), Hostile(z_1)\}$
$q' \leftarrow \texttt{Subst}(\{x_1/West, x_2/y_1, y_1/M_1\}, Sells(x_1, y_1, z_1))$
$\theta' \leftarrow \{x_3/M_1, z_1/Nono\}$
$new\_goals \leftarrow \{Missile(x_3), Owns(Nono, x_3), Hostile(z_1)\}$

Used clause (after Standardize-Apart):
$Missile(x_3) \wedge Owns(Nono, x_3) \Rightarrow Sells(West, x_3, Nono)$

Criminal West: Backward Chaining (6)

function FOL-BC-Ask (KB,$goals$,θ) **returns** a set of substitutions

if $goals$ is empty **then return** $\{\theta\}$
$q' \leftarrow \texttt{Subst}(\theta, \texttt{First}(goals))$
for each sentence r **in** KB where $\texttt{Standardize-Apart}(r) = (p_1 \wedge \ldots \wedge p_n \Rightarrow q)$ and $\theta' \leftarrow \texttt{Unify}(q, q')$ succeeds
  $new\_goals \leftarrow [p_1, \ldots, p_n | \texttt{Rest}(goals)]$
  $answers \leftarrow \texttt{FOL-BC-Ask}(KB, new\_goals, \texttt{Compose}(\theta', \theta)) \cup answers$
return $answers$



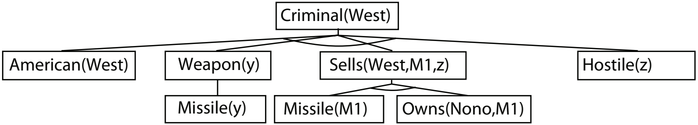
$goals$: $\{Missile(x_3), Owns(Nono, x_3), Hostile(z_1)\}$
$q' \leftarrow \texttt{Subst}(\{x_1/West, x_2/y_1, y_1/M_1, x_3/M_1, z_1/Nono\},\ Missile(x_3))$
$\theta' \leftarrow \emptyset$
$new\_goals \leftarrow \{Owns(Nono, x_3), Hostile(z_1)\}$

Used clause (after Standardize-Apart):
$Missile(M_1)$

Criminal West: Backward Chaining (7)

function FOL-BC-Ask (KB,$goals$,θ) **returns** a set of substitutions

if $goals$ is empty **then return** $\{\theta\}$
$q' \leftarrow \texttt{Subst}(\theta, \texttt{First}(goals))$
for each sentence r **in** KB where $\texttt{Standardize-Apart}(r) = (p_1 \wedge \ldots \wedge p_n \Rightarrow q)$ and $\theta' \leftarrow \texttt{Unify}(q, q')$ succeeds
    $new\_goals \leftarrow [p_1, \ldots, p_n | \texttt{Rest}(goals)]$
    $answers \leftarrow \texttt{FOL-BC-Ask}(KB, new\_goals, \texttt{Compose}(\theta', \theta)) \cup answers$
return $answers$



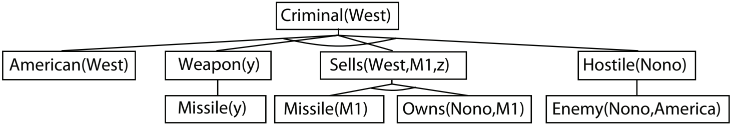
$goals$: $\{Owns(Nono, x_3), Hostile(z_1)\}$
$q' \leftarrow \texttt{Subst}(\{x_1/West, x_2/y_1, y_1/M_1, x_3/M_1, z_1/Nono\}, Owns(Nono, x_3))$
$\theta' \leftarrow \varnothing$
$new\_goals \leftarrow \{Hostile(z_1)\}$

Used clause (after Standardize-Apart):
$Owns(Nono, M_1)$

Criminal West: Backward Chaining (8)

function FOL-BC-Ask (KB,$goals$,θ) **returns** a set of substitutions

if $goals$ is empty **then return** $\{\theta\}$
$q' \leftarrow \texttt{Subst}(\theta, \texttt{First}(goals))$
for each sentence r **in** KB where $\texttt{Standardize-Apart}(r) = (p_1 \wedge \ldots \wedge p_n \Rightarrow q)$ and $\theta' \leftarrow \texttt{Unify}(q, q')$ succeeds
    $new\_goals \leftarrow [p_1, \ldots, p_n | \texttt{Rest}(goals)]$
    $answers \leftarrow \texttt{FOL-BC-Ask}(KB, new\_goals, \texttt{Compose}(\theta', \theta)) \cup answers$
return $answers$



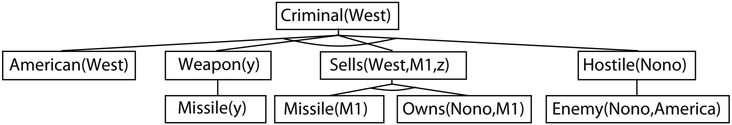
$goals$: $\{Hostile(z_1)\}$
$q' \leftarrow \texttt{Subst}(\{x_1/West, x_2/y_1, y_1/M_1, x_3/M_1, z_1/Nono\}, Hostile(z_1))$
$\theta' \leftarrow \{x_4/Nono\}$
$new\_goals \leftarrow \{Enemy(x_4, America)\}$

Used clause (after Standardize-Apart):
$Enemy(x_4, America) \Rightarrow Hostile(x_4)$

Criminal West: Backward Chaining (9)

function FOL-BC-Ask (KB,$goals$,θ) **returns** a set of substitutions

if *goals* is empty **then return** $\{\theta\}$
$q' \leftarrow \texttt{Subst}(\theta, \texttt{First}(goals))$
for each sentence r **in** KB where $\texttt{Standardize-Apart}(r) = (p_1 \wedge \ldots \wedge p_n \Rightarrow q)$ and $\theta' \leftarrow \texttt{Unify}(q, q')$ succeeds
  $new\_goals \leftarrow [p_1, \ldots, p_n | \texttt{Rest}(goals)]$
  $answers \leftarrow \texttt{FOL-BC-Ask}(KB, new\_goals, \texttt{Compose}(\theta', \theta)) \cup answers$
return *answers*



$goals$: $\{Enemy(x_4, America)\}$
$q' \leftarrow \texttt{Subst}(\{x_1/West, x_2/y_1, y_1/M_1, x_3/M_1, z_1/Nono, x_4/Nono\}, Enemy(x_4, America))$
$\theta' \leftarrow \emptyset$
$new\_goals \leftarrow \emptyset$

Used clause (after Standardize-Apart):
$Enemy(Nono, America)$

Properties of Backward Chaining

- Depth-first recursive proof search: space is linear in size of proof.
- Incomplete due to infinite loops
  $\Rightarrow$ fix by checking current goal against every goal on stack.
- Inefficient due to repeated subgoals (both success and failure)
  $\Rightarrow$ fix using caching of previous results (extra space!).
- Widely used for **logic programming**.

Logic Programming

Logic programming comes fairly close to the idea of solving problems by running inference processes on a knowledge base.

	Logic programming	**Ordinary programming**
1.	Identify problem	Identify problem
2.	Assemble information	Assemble information
3.	Tea break	Figure out solution
4.	Encode information in KB	Program solution
5.	Encode problem instance as facts	Encode problem instance as data
6.	Ask queries	Apply program to data
7.	Find false facts	Debug procedural errors

Should be easier to debug $Capital(NewYork, US)$ than $x := x + 2$.

Prolog

- Basis: backward chaining with Horn clauses.
- Primarily used as rapid-prototyping language and for symbol-manipulation tasks such as writing compilers.

Prolog syntax

Different notation compared to the conventions in logic.

- Uppercase letters for variables, lowercase letters for constants (exactly the opposite to our convention)
- Commas separate conjuncts in a clause, and the clause is written "backwards"; instead of $A \wedge B \Rightarrow C$ we have $C :- A, B$ in Prolog. Example:

  ```
  criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
  ```

- [E|L] denotes a list whose first element is E and whose rest is L.

appending E to list L

Prolog: Example

We design a short Prolog program for appending lists X and Y to a list Z:

```
append([],Y,Y).
append([A|X],Y,[A|Z]) :- append(X,Y,Z).
```

In English:

- Appending an empty list with a list Y produces the same list Y.
- Given that Z is the result of appending X onto Y, we receive [A|Z] when appending [A|X] onto Y.

The Prolog implementation is actually more powerful than in procedural programming languages (such as C); e.g., we can query how two lists can be appended to $[1,2]$ using append(X,Y,[1,2]):

```
X=[]      Y=[1,2];
X=[1]     Y=[2];
X=[1,2]   Y=[]
```

Tweedback Question

What is this program doing?

```
a(X,X):- X >= 0.
a(X,Y):- Y is -X.
```

A Increments each number.
B Compute the absolute value.
C Count characters in a word.

how program works

```
?- a(0,R).
R = 0
?- a(-9,R).
R = 9
?- a(-9,9).
yes
?- a(-9,8).
no
?- a(I,8).
I = 8
```

Resolution in First-Order Logic

- Resolution in propositional logic enabled a complete proof procedure in propositional logic.
- We try to achieve the same for first-order logic.
- As for first-order logic, we first have to convert our sentence to conjunctive normal form (CNF).
- Next, we present the resolution inference rule for first-order logic.
- Finally, we discuss the completeness of the resolution procedure.

Conjunctive Normal Form for First-Order Logic

Important

Conjunctive normal form is identical to the one in propositional logic, except that literals are allowed to be universally quantified variables.

Example

$\forall x \quad , \forall y \quad , \forall z$

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

can be written in CNF using *implication elimination* and *De Morgan* as

$\forall x \quad , \forall y \quad , \forall z$

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

Every sentence of first-order logic can be converted into an inferentially equivalent CNF sentence as shown on the next slides.

Conversion to CNF (1)

The main difference to propositional logic is the need to eliminate existential quantifiers.

Running example

"Everyone who loves all animals is loved by someone":

$$\forall x \quad [\forall y \quad Animal(y) \Rightarrow Loves(x, y)] \Rightarrow [\exists y \quad Loves(y, x)].$$

1 - **Eliminate implications**

  Replace $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$:

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

2 - **Move $\neg$ inwards**

  In addition to the rules for propositional logic, we need rules for negated quantifiers:

$$\neg\forall x \quad p \text{ becomes } \quad \exists x \quad \neg p$$

$$\neg\exists x \quad p \text{ becomes } \quad \forall x \quad \neg p$$

Conversion to CNF (2)

Our sentence goes through the following transformations:

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

2 different ys!

3 - **Standardize variables**
  For sentences like $(\exists x \quad P(x)) \vee (\exists x \quad Q(x))$ which use the same variable name twice, change the name of one of the variables to avoid confusion when dropping quantifiers:

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

Conversion to CNF (3)

4 • **Skolemization**
Skolemization is the process of removing existential quantifiers. In the simple case, it is equal to the Existential Instantiation rule on slide 6: translate $\exists x \quad P(x)$ into $P(A)$ and introduce A as a new constant. If we apply this simple rule to our example (which does not have the form $\exists x \quad P(x)$) we get

$$\forall x \quad [Animal(A) \land \neg Loves(x, A)] \lor Loves(B, x)$$

which has the wrong meaning: it says that everyone either fails to love a particular animal A or is loved by some particular entity B. We fix this by introducing **Skolem functions** F and G:

$$\forall x \quad [Animal(F(x)) \land \neg Loves(x, F(x))] \lor Loves(G(x), x)$$

The arguments of the Skolem function are universally quantified. The quantifiers of these arguments precede that of the existentially quantified variable.

Conversion to CNF (4)

5 • **Drop universal quantifiers**
At this point we have only universal quantifiers. Since same quantifiers can be moved to the left, we can drop them and only assume them from now on.

$$[Animal(F(x)) \land \neg Loves(x, F(x))] \lor Loves(G(x), x)$$

6 • **Distribute $\lor$ over $\land$**

$$[Animal(F(x)) \lor Loves(G(x), x)] \land$$
$$[\neg Loves(x, F(x)) \lor Loves(G(x), x)]$$

The sentence is now in CNF.

Resolution Inference Rule

Important

The resolution rule of propositional logic can be lifted to first-order logic:

Resolution rule for first-order logic

$$\frac{l_1 \vee \ldots \vee l_k, \quad m_1 \vee \ldots \vee m_n}{\texttt{Subst}(\theta, l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)},$$

where $\texttt{Unify}(l_i, \neg m_j) = \theta$.

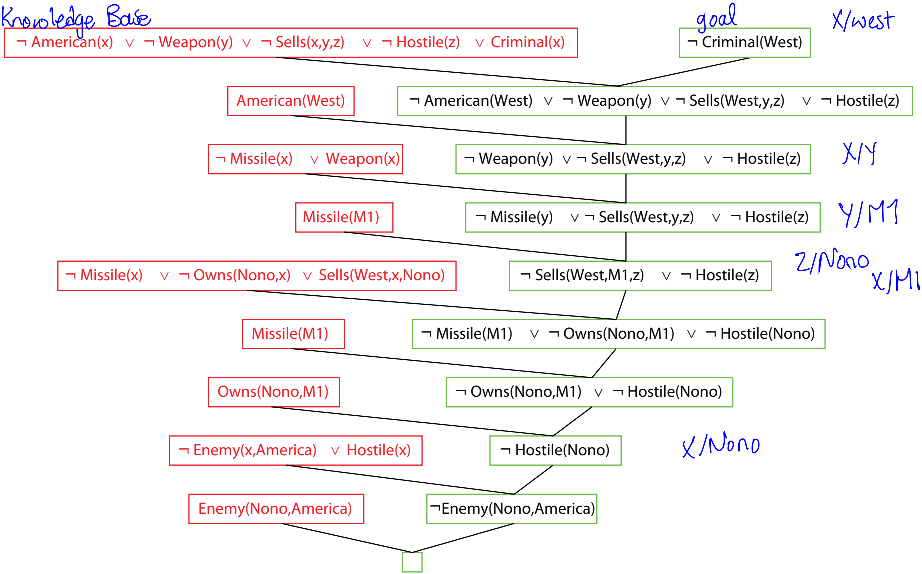
Example: We can resolve the two clauses

$$[Animal(F(x)) \vee Loves(G(x), x)] \text{ and } [\neg Loves(u, v) \vee \neg Kills(u, v)]$$

by eliminating the complementary literals $Loves(G(x), x)$ and $\neg Loves(u, v)$, with unifier $\theta = \{u/G(x), v/x\}$, to produce the **resolvent** clause

$$[Animal(F(x)) \vee \neg Kills(G(x), x)].$$

Proof by Resolution: Example



Proof by Resolution: Another Example (I)

Given is the following knowledge base:

- **Everyone who loves all animals is loved by someone (see next slide)**:
  $\forall x \quad [\forall y \quad Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y \quad Loves(y,x)]$
  CNF_1: $Animal(F(x)) \vee Loves(G(x),x)$
  CNF_2: $\neg Loves(x,F(x)) \vee Loves(G(x),x)$
- **Anyone who kills an animal is loved by no one**:
  $\forall x \quad [\exists z \quad Animal(z) \wedge Kills(x,z)] \Rightarrow [\forall y \quad \neg Loves(y,x)]$
  CNF: $\neg Loves(y,x) \vee \neg Animal(z) \vee \neg Kills(x,z)$
- **Jack loves all animals**:
  $\forall x \quad Animal(x) \Rightarrow Loves(Jack,x)$
  CNF: $\neg Animal(x) \vee Loves(Jack,x)$
- **Either Jack or Curiosity killed the cat, who is named Tuna**:
  $Kills(Jack,Tuna) \vee Kills(Curiosity,Tuna)$
  $Cat(Tuna)$
- **Required background knowledge**:
  $\forall x \quad Cat(x) \Rightarrow Animal(x)$
  CNF: $\neg Cat(x) \vee Animal(x)$

Proof by Resolution: Another Example (II)

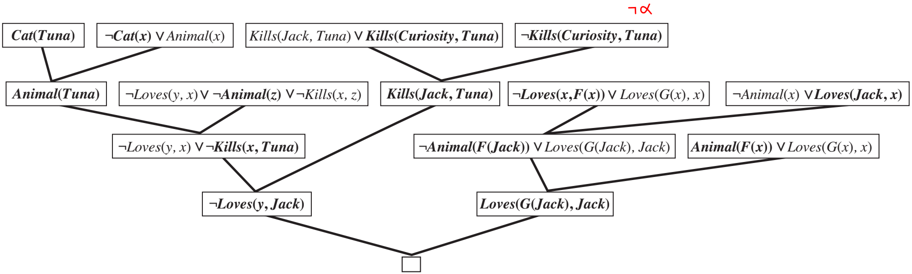
- **We show the conversion into CNF for**:
  $\forall x \quad [\forall y \quad Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y \quad Loves(y,x)]$
- $\mathbf{A} \Rightarrow \mathbf{B} \equiv \neg \mathbf{A} \vee \mathbf{B}$:
  $\forall x \quad [\neg \forall y \quad Animal(y) \Rightarrow Loves(x,y)] \vee [\exists y \quad Loves(y,x)]$
- $\mathbf{A} \Rightarrow \mathbf{B} \equiv \neg \mathbf{A} \vee \mathbf{B}$:
  $\forall x \quad [\neg \forall y \quad \neg Animal(y) \vee Loves(x,y)] \vee [\exists y \quad Loves(y,x)]$
- $\neg \forall \mathbf{x} \quad \mathbf{P}(\mathbf{x}) \equiv \exists \mathbf{x} \quad \neg \mathbf{P}(\mathbf{x})$:
  $\forall x \quad [\exists y \quad \neg(\neg Animal(y) \vee Loves(x,y))] \vee [\exists y \quad Loves(y,x)]$
- $\neg(\mathbf{A} \vee \mathbf{B}) \equiv \neg \mathbf{A} \wedge \neg \mathbf{B}$:
  $\forall x \quad [\exists y \quad Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y \quad Loves(y,x)]$
- **Skolemization**:
  $\forall x \quad [Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee [Loves(G(x), x)]$
- $(\mathbf{A} \wedge \mathbf{B}) \vee \mathbf{C} \equiv (\mathbf{A} \vee \mathbf{C}) \wedge (\mathbf{B} \vee \mathbf{C})$:
  $\forall x \quad [Animal(F(x)) \vee Loves(G(x), x)] \wedge [\neg Loves(x, F(x)) \vee Loves(G(x), x)]$

Proof by Resolution: Another Example (III)

We are interested in the question “Did Curiosity kill the cat?”, which is formalized as

$$\alpha = Kills(Curiosity, Tuna).$$

To show $KB \models \alpha$, we show that $KB \wedge \neg\alpha$ is unsatisfiable:



Crime Report

In order to write the crime report, we can use the proof and translate the individual steps in natural language:

Report

Suppose Curiosity did not kill Tuna. We know that either Jack or Curiosity did; thus Jack must have. Now, Tuna is a cat and cats are animals, so Tuna is an animal. Because anyone who kills an animal is loved by no one, we know that no one loves Jack. On the other hand, Jack loves all animals, so someone loves him; so we have a contradiction. Therefore, Curiosity killed the cat.

Completeness of Resolution in First-Order Logic

- In propositional logic, we can prove that resolution provides a complete proof procedure.
- In first-order logic, we can show that resolution can always prove that a sentence is unsatisfiable. We say resolution in first-order logic is **refutation-complete**.
- Attempting to prove a satisfiable first-order formula as unsatisfiable may result in a nonterminating computation, which cannot happen in propositional logic. In FOL, you can compute infinitely
- Since we have already proven the completeness for propositional logic, we skip the proof for refutation-completeness in first-order logic.

Equality

- So far we have not considered any technique that can handle equality, such as $x = y$.
- Sentences involving equality are hard to handle, so we skip this aspect.
- Some methods on dealing with equality are described in Sec. 5.5 of the chapter *Inference in First Order Logic* in the AI book.

Summary

- **Propositionalization** is a possibility to use inference rules from propositional logic on a first-order logic sentence. However, this generates a vast number of propositional sentences.
- **Generalized Modus Ponens** is a powerful inference rule that makes it possible to use forward-chaining and backward-chaining of Horn clauses in first-order logic.
- Generalized Modus Ponens is complete for Horn clauses, although the entailment problem is semidecidable. For **Datalog** knowledge bases (no functions), entailment is decidable.
- The generalized **resolution** inference rule provides a complete proof system for first-order logic, using knowledge bases in conjunctive normal form.

* Logic plays critical role in General AI