Inference in First-Order Logic

Philipp Koehn

12 March 2019
<table>
<thead>
<tr>
<th>Year</th>
<th>Person</th>
<th>Logic/Inference</th>
</tr>
</thead>
<tbody>
<tr>
<td>450B.C.</td>
<td>Stoics</td>
<td>propositional logic, inference (maybe)</td>
</tr>
<tr>
<td>322B.C.</td>
<td>Aristotle</td>
<td>“syllogisms” (inference rules), quantifiers</td>
</tr>
<tr>
<td>1565</td>
<td>Cardano</td>
<td>probability theory (propositional logic + uncertainty)</td>
</tr>
<tr>
<td>1847</td>
<td>Boole</td>
<td>propositional logic (again)</td>
</tr>
<tr>
<td>1879</td>
<td>Frege</td>
<td>first-order logic</td>
</tr>
<tr>
<td>1922</td>
<td>Wittgenstein</td>
<td>proof by truth tables</td>
</tr>
<tr>
<td>1930</td>
<td>Gödel</td>
<td>$\exists$ complete algorithm for FOL</td>
</tr>
<tr>
<td>1930</td>
<td>Herbrand</td>
<td>complete algorithm for FOL (reduce to propositional)</td>
</tr>
<tr>
<td>1931</td>
<td>Gödel</td>
<td>$\neg\exists$ complete algorithm for arithmetic</td>
</tr>
<tr>
<td>1960</td>
<td>Davis/Putnam</td>
<td>“practical” algorithm for propositional logic</td>
</tr>
<tr>
<td>1965</td>
<td>Robinson</td>
<td>“practical” algorithm for FOL—resolution</td>
</tr>
</tbody>
</table>
The Story So Far

- Propositional logic
- Subset of prepositional logic: horn clauses
- Inference algorithms
  - forward chaining
  - backward chaining
  - resolution (for full prepositional logic)
- First order logic (FOL)
  - variables
  - functions
  - quantifiers
  - etc.
- Today: inference for first order logic
Outline

- Reducing first-order inference to propositional inference
- Unification
- Generalized Modus Ponens
- Forward and backward chaining
- Logic programming
- Resolution
reduction to

prepositional inference
Universal Instantiation

- Every instantiation of a universally quantified sentence is entailed by it:

\[ \forall v \alpha \]

Substituting \( \{v/g\}, \alpha \)

for any variable \( v \) and ground term \( g \)

- E.g., \( \forall x \ King(x) \land Greedy(x) \rightarrow Evil(x) \) yields

\[
\begin{align*}
  King(John) \land Greedy(John) & \rightarrow Evil(John) \\
  King(Richard) \land Greedy(Richard) & \rightarrow Evil(Richard) \\
  King(Father(John)) \land Greedy(Father(John)) & \rightarrow Evil(Father(John))
\end{align*}
\]

\[ \vdots \]
Existential Instantiation

• For any sentence $\alpha$, variable $v$, and constant symbol $k$ that does not appear elsewhere in the knowledge base:

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

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

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

provided $C_1$ is a new constant symbol, called a Skolem constant
Instantiation

- Universal Instantiation
  - can be applied several times to **add** new sentences
  - the new KB is logically equivalent to the old

- Existential Instantiation
  - can be applied once to **replace** the existential sentence
  - the new KB is **not** equivalent to the old
  - but is satisfiable iff the old KB was satisfiable
Reduction to Propositional Inference

- Suppose the KB contains just the following:
  \[\forall x \ King(x) \land Greedy(x) \implies Evil(x)\]
  
  \[\begin{align*}
  & King(John) \\
  & Greedy(John) \\
  & Brother(Richard, John)
  \end{align*}\]

- Instantiating the universal sentence in **all possible** ways, we have:
  \[\begin{align*}
  & King(John) \land Greedy(John) \implies Evil(John) \\
  & King(Richard) \land Greedy(Richard) \implies Evil(Richard) \\
  & King(John) \\
  & Greedy(John) \\
  & Brother(Richard, John)
  \end{align*}\]

- The new KB is **propositionalized**: proposition symbols are
  \[\begin{align*}
  & King(John), Greedy(John), Evil(John), Brother(Richard, John), \text{etc.}
  \end{align*}\]
Reduction to Propositional Inference

- Claim: a ground sentence \( \ast \) is entailed by new KB iff entailed by original KB
- 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., \( \text{Father} (\text{Father} (\text{Father} (\text{John}))) \)
- Theorem: Herbrand (1930). If a sentence \( \alpha \) is entailed by an FOL KB, it is entailed by a finite subset of the propositional KB
- Idea: For \( n = 0 \) to \( \infty \) do
  create a propositional KB by instantiating with depth-\( n \) terms
  see if \( \alpha \) is entailed by this KB
- Problem: works if \( \alpha \) is entailed, loops if \( \alpha \) is not entailed
- Theorem: Turing (1936), Church (1936), entailment in FOL is semidecidable
Practical Problems with Propositionalization

- Propositionalization seems to generate lots of irrelevant sentences.

- E.g., from

\[ \forall x \ King(x) \land Greedy(x) \implies Evil(x) \]
\[ King(\text{John}) \]
\[ \forall y \ Greedy(y) \]
\[ Brother(\text{Richard, John}) \]

it seems obvious that \text{Evil(John)}, but propositionalization produces lots of facts such as \text{Greedy(Richard)} that are irrelevant

- With \( p \) \( k \)-ary predicates and \( n \) constants, there are \( p \cdot n^k \) instantiations

- With function symbols, it gets much much worse!
unification
Plan

• We have the inference rule
  \[ \forall x \ King(x) \land Greedy(x) \implies Evil(x) \]

• We have facts that (partially) match the precondition
  - \( King(John) \)
  - \( \forall y \ Greedy(y) \)

• We need to match them up with substitutions: \( \theta = \{ x/John, y/John \} \) works
  - unification
  - generalized modus ponens
Unification

- **Unify**$(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$\theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knows(John, $x$)</td>
<td>Knows(John, Jane)</td>
<td></td>
</tr>
<tr>
<td>Knows(John, $x$)</td>
<td>Knows($y$, Mary)</td>
<td></td>
</tr>
<tr>
<td>Knows(John, $x$)</td>
<td>Knows($y$, Mother($y$))</td>
<td></td>
</tr>
<tr>
<td>Knows(John, $x$)</td>
<td>Knows($x$, Mary)</td>
<td></td>
</tr>
</tbody>
</table>
**Unification**

- **UNIFY**($\alpha, \beta$) = $\theta$ if $\alpha \theta = \beta \theta$

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$\theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knows(John, x)</td>
<td>Knows(John, Jane)</td>
<td>${x/Jane}$</td>
</tr>
<tr>
<td>Knows(John, x)</td>
<td>Knows(y, Mary)</td>
<td></td>
</tr>
<tr>
<td>Knows(John, x)</td>
<td>Knows(y, Mother(y))</td>
<td></td>
</tr>
<tr>
<td>Knows(John, x)</td>
<td>Knows(x, Mary)</td>
<td></td>
</tr>
</tbody>
</table>
### Unification

- **Unify**($\alpha, \beta$) = $\theta$ if $\alpha\theta = \beta\theta$

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$\theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Knows}(\text{John}, x)$</td>
<td>$\text{Knows}(\text{John}, \text{Jane})$</td>
<td>${x/\text{Jane}}$</td>
</tr>
<tr>
<td>$\text{Knows}(\text{John}, x)$</td>
<td>$\text{Knows}(y, \text{Mary})$</td>
<td>${x/\text{Mary}, y/\text{John}}$</td>
</tr>
<tr>
<td>$\text{Knows}(\text{John}, x)$</td>
<td>$\text{Knows}(y, \text{Mother}(y))$</td>
<td></td>
</tr>
<tr>
<td>$\text{Knows}(\text{John}, x)$</td>
<td>$\text{Knows}(x, \text{Mary})$</td>
<td></td>
</tr>
</tbody>
</table>
Unification

- \textbf{UNIFY}(\alpha, \beta) = \theta \text{ if } \alpha \theta = \beta \theta

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>\theta</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knows(John, x)</td>
<td>Knows(John, Jane)</td>
<td>{x/Jane}</td>
</tr>
<tr>
<td>Knows(John, x)</td>
<td>Knows(y, Mary)</td>
<td>{x/Mary, y/John}</td>
</tr>
<tr>
<td>Knows(John, x)</td>
<td>Knows(y, Mother(y))</td>
<td>{y/John, x/Mother(John)}</td>
</tr>
<tr>
<td>Knows(John, x)</td>
<td>Knows(x, Mary)</td>
<td></td>
</tr>
</tbody>
</table>
Unification

- $\text{UNIFY}(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$\theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Knows}(\text{John}, x)$</td>
<td>$\text{Knows}(\text{John}, \text{Jane})$</td>
<td>${x/\text{Jane}}$</td>
</tr>
<tr>
<td>$\text{Knows}(\text{John}, x)$</td>
<td>$\text{Knows}(y, \text{Mary})$</td>
<td>${x/\text{Mary}, y/\text{John}}$</td>
</tr>
<tr>
<td>$\text{Knows}(\text{John}, x)$</td>
<td>$\text{Knows}(y, \text{Mother}(y))$</td>
<td>${y/\text{John}, x/\text{Mother}(\text{John})}$</td>
</tr>
<tr>
<td>$\text{Knows}(\text{John}, x)$</td>
<td>$\text{Knows}(x, \text{Mary})$</td>
<td>fail</td>
</tr>
</tbody>
</table>

- **Standardizing apart** eliminates overlap of variables, e.g., $\text{Knows}(z_{17}, \text{Mary})$

  $\text{Knows}(\text{John}, x) \land \text{Knows}(z_{17}, \text{Mary}) \lor \{z_{17}/\text{John}, x/\text{Mary}\}$
generalized modus ponens
Generalized Modus Ponens

- Generalized modus ponens used with KB of **definite clauses** (exactly one positive literal)

- All variables assumed universally quantified

\[
p_1', p_2', \ldots, p_n', (p_1 \land p_2 \land \ldots \land p_n \Rightarrow q) \quad \frac{q\theta}{p_i'\theta = p_i\theta \text{ for all } i}
\]

- Rule: \( King(x) \land Greedy(x) \implies Evil(x) \)

- Precondition of rule: \( p_1 \text{ is } King(x) \quad p_2 \text{ is } Greedy(x) \)

- Implication: \( q \text{ is } Evil(x) \)

- Facts: \( p_1' \text{ is } King(John) \quad p_2' \text{ is } Greedy(y) \)

- Substitution: \( \theta \text{ is } \{x/John, y/John\} \)

\( \Rightarrow \) Result of modus ponens: \( q\theta \text{ is } Evil(John) \)
forward chaining
Example Knowledge

- 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 a criminal
Example Knowledge Base

- ... it is a crime for an American to sell weapons to hostile nations:
  \[ \text{American}(x) \land \text{Weapon}(y) \land \text{Sells}(x, y, z) \land \text{Hostile}(z) \implies \text{Criminal}(x) \]

- Nono ... has some missiles, i.e., \( \exists x \ \text{Owns}(\text{Nono}, x) \land \text{Missile}(x) \):
  \[ \text{Owns}(\text{Nono}, M_1) \text{ and } \text{Missile}(M_1) \]

- ... all of its missiles were sold to it by Colonel West:
  \[ \forall x \ \text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \implies \text{Sells}(\text{West}, x, \text{Nono}) \]

- Missiles are weapons:
  \[ \text{Missile}(x) \implies \text{Weapon}(x) \]

- An enemy of America counts as “hostile”:
  \[ \text{Enemy}(x, \text{America}) \implies \text{Hostile}(x) \]

- West, who is American ...
  \[ \text{American}(\text{West}) \]

- The country Nono, an enemy of America ...
  \[ \text{Enemy}(\text{Nono}, \text{America}) \]
Forward Chaining Proof

\begin{tabular}{llll}
  \textbf{American(West)} & \textbf{Missile(M1)} & \textbf{Owns(Nono,M1)} & \textbf{Enemy(Nono,America)} \\
\end{tabular}
Forward Chaining Proof

(Note: $\forall x \text{ Missile}(x) \land \text{Owns}(\text{Nono}, x) \implies \text{Sells}(\text{West}, x, \text{Nono})$)
Forward Chaining Proof

(Note: $\text{American}(x) \land \text{Weapon}(y) \land \text{Sells}(x, y, z) \land \text{Hostile}(z) \implies \text{Criminal}(x)$)
Properties of Forward Chaining

- Sound and complete for first-order definite clauses (proof similar to propositional proof)

- **Datalog** (1977) = first-order definite clauses + no functions (e.g., crime example)
  Forward chaining terminates for Datalog in poly iterations: at most $p \cdot n^k$ literals

- May not terminate in general if $\alpha$ is not entailed

- This is unavoidable: entailment with definite clauses is semidecidable
Efficiency of Forward Chaining

- Simple observation: no need to match a rule on iteration $k$ if a premise wasn’t added on iteration $k - 1$
  $\implies$ match each rule whose premise contains a newly added literal

- Matching itself can be expensive

- **Database indexing** allows $O(1)$ retrieval of known facts
  e.g., query $\text{Missile}(x)$ retrieves $\text{Missile}(M_1)$

- Matching conjunctive premises against known facts is NP-hard

- Forward chaining is widely used in **deductive databases**
Hard Matching Example

\[ \text{Diff}(wa, nt) \land \text{Diff}(wa, sa) \land \]
\[ \text{Diff}(nt, q) \land \text{Diff}(nt, sa) \land \]
\[ \text{Diff}(q, nsw) \land \text{Diff}(q, sa) \land \]
\[ \text{Diff}(nsw, v) \land \text{Diff}(nsw, sa) \land \]
\[ \text{Diff}(v, sa) \implies \text{Colorable}() \]
\[ \text{Diff}(\text{Red}, \text{Blue}) \land \text{Diff}(\text{Red}, \text{Green}) \]
\[ \text{Diff}(\text{Green}, \text{Red}) \land \text{Diff}(\text{Green}, \text{Blue}) \]
\[ \text{Diff}(\text{Blue}, \text{Red}) \land \text{Diff}(\text{Blue}, \text{Green}) \]

- \text{Colorable}() is inferred iff the constraint satisfaction problem has a solution

- CSPs include 3SAT as a special case, hence matching is NP-hard
Forward Chaining Algorithm

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 (p₁ ∧ ... ∧ pₙ)θ = (p'₁ ∧ ... ∧ p'ₙ)θ
            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
backward chaining
Backward Chaining

- Start with query

- Check if it can be derived by given rules and facts
  - apply rules that infer the query
  - recurse over pre-conditions
Backward Chaining Example

\[\text{Criminal(West)}\]
Backward Chaining Example

Diagram:
- Criminal(West)
  - {x/West}
  - American(x)
  - Weapon(y)
  - Sells(x,y,z)
  - Hostile(z)
Backward Chaining Example

Diagram:

- Criminal(West)
- American(West)
- Weapon(y)
- Sells(x,y,z)
- Hostile(z)
- \{x/West\}

{ }
Backward Chaining Example

```
American(West) { } Weapon(y) Sells(x,y,z) Hostile(z)
       /         |               |
      /           |               |
Criminal(West)       {x/West}       Missile(y)
```

Philipp Koehn
Artificial Intelligence: Inference in First-Order Logic
12 March 2019
Backward Chaining Example

```
American(West)
{ }

Weapon(y)

Sells(x,y,z)

Hostile(z)

Criminal(West)
{x/West, y/M1}

Missile(y)
{ y/M1 }
```
Backward Chaining Example

\[
\text{Criminal}(West) \\
\text{American}(West) \quad \text{Weapon}(y) \quad \text{Sells}(West,M1,z) \quad \text{Hostile}(z)
\]

\[
\{ x/\text{West}, y/M1, z/\text{None} \}
\]

\[
\text{Missile}(y) \quad \text{Missile}(M1) \quad \text{Owns}(\text{None},M1)
\]

\[
\{ y/M1 \}
\]
Backward Chaining Example

\[
\text{Criminal}(\text{West}) \quad \{x/\text{West}, y/M1, z/\text{Nono}\}
\]

- \text{American}(\text{West})
- \text{Weapon}(y) \quad \{y/M1\}
- \text{Sells}(\text{West}, M1, z) \quad \{z/\text{Nono}\}
- \text{Hostile}(\text{Nono})

- \text{Missile}(y) \quad \{y/M1\}
- \text{Missile}(M1) \quad \{\}\n- \text{Owns}(\text{Nono}, M1) \quad \{\}\n- \text{Enemy}(\text{Nono}, \text{America}) \quad \{\}\n
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 (without improvements!) for logic programming
Backward Chaining Algorithm

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

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

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

if goals is empty then return \{ \( \theta \) \}

\( q' \leftarrow \text{SUBST}(\theta, \text{FIRST}(goals)) \)

for each sentence \( r \) in KB

where \( \text{STANDARDIZE-Apart}(r) = (p_1 \land \ldots \land p_n \Rightarrow q) \)
and \( \theta' \leftarrow \text{UNIFY}(q, q') \) succeeds

\( \text{new_goals} \leftarrow [p_1, \ldots, p_n | \text{REST}(goals)] \)
\( \text{answers} \leftarrow \text{FOL-BC-Ask}(KB, \text{new_goals}, \text{COMPOSE}(\theta', \theta)) \cup \text{answers} \)

return \text{answers}
logic programming
Logic Programming

- Sound bite: computation as inference on logical KBs

  Logic programming
  1. Identify problem
  2. Assemble information
  3. Tea break
  4. Encode information in KB
  5. Encode problem instance as facts
  6. Ask queries
  7. Find false facts

  Ordinary programming
  Identify problem
  Assemble information
  Figure out solution
  Program solution
  Encode problem instance as data
  Apply program to data
  Debug procedural errors

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

- Basis: backward chaining with Horn clauses + bells & whistles

- Widely used in Europe, Japan (basis of 5th Generation project)

- Compilation techniques ⇒ approaching a billion logical inferences per second

- Program = set of clauses = head :- literal\(_1\), ... literal\(_n\).

```
criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
missile(M\(_1\)).
owns(Nono,M\(_1\)).
sells(West,X,Nono) :- missile(X), owns(Nono,X).
weapon(X) :- missile(X).
hostile(X) :- enemy(X,America).
American(West).
Enemy(Nono,America).
```
Prolog Systems

- Efficient unification by open coding
- Efficient retrieval of matching clauses by direct linking
- Depth-first, left-to-right backward chaining
- Built-in predicates for arithmetic etc., e.g., $X \text{ is } Y*Z+3$
- Closed-world assumption ("negation as failure")
  e.g., given $\text{alive}(X) :\neg \text{dead}(X)$.
  $\text{alive}(\text{joe})$ succeeds if $\text{dead}(\text{joe})$ fails
resolution
Resolution: Brief Summary

- Full first-order version:

\[
\begin{align*}
\ell_1 \lor \cdots \lor \ell_k, & \quad m_1 \lor \cdots \lor m_n \\
\left( \ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n \right) \theta
\end{align*}
\]

where \( \text{UNIFY}(\ell_i, \neg m_j) = \theta \).

- For example,

\[
\begin{array}{l}
\neg \text{Rich}(x) \lor \text{Unhappy}(x) \\
\text{Rich}(\text{Ken})
\end{array}
\]

\[
\begin{array}{l}
\frac{}{\text{Unhappy}(\text{Ken})}
\end{array}
\]

with \( \theta = \{x/\text{Ken}\} \)

- Apply resolution steps to \( \text{CNF}(KB \land \neg \alpha) \); complete for FOL
Conversion to CNF

Everyone who loves all animals is loved by someone:
\[ \forall x \left[ \forall y \text{Animal}(y) \implies \text{Loves}(x,y) \right] \implies \left[ \exists y \text{Loves}(y,x) \right] \]

1. Eliminate biconditionals and implications

\[ \forall x \left[ \neg \forall y \neg \text{Animal}(y) \lor \text{Loves}(x,y) \right] \lor \left[ \exists y \text{Loves}(y,x) \right] \]

2. Move \( \neg \) inwards: \( \neg \forall x, p \equiv \exists x \neg p \), \( \neg \exists x, p \equiv \forall x \neg p \):

\[ \forall x \left[ \exists y \neg (\neg \text{Animal}(y) \lor \text{Loves}(x,y)) \right] \lor \left[ \exists y \text{Loves}(y,x) \right] \]
\[ \forall x \left[ \exists y \neg \text{Animal}(y) \land \neg \text{Loves}(x,y) \right] \lor \left[ \exists y \text{Loves}(y,x) \right] \]
\[ \forall x \left[ \exists y \text{Animal}(y) \land \neg \text{Loves}(x,y) \right] \lor \left[ \exists y \text{Loves}(y,x) \right] \]
Conversion to CNF

3. Standardize variables: each quantifier should use a different one

\[ \forall x [\exists y \ Animal(y) \land \neg Loves(x, y)] \lor [\exists z \ Loves(z, x)] \]

4. Skolemize: a more general form of existential instantiation. Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:

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

5. Drop universal quantifiers:

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

6. Distribute \( \land \) over \( \lor \):

\[ [Animal(F(x)) \lor Loves(G(x), x)] \land [\neg Loves(x, F(x)) \lor Loves(G(x), x)] \]
Our Previous Example

• Rules
  – American(x) ∧ Weapon(y) ∧ Sells(x, y, z) ∧ Hostile(z) → Criminal(x)
  – Missile(M₁) and Owns(Nono, M₁)
  – ∀ x  Missile(x) ∧ Owns(Nono, x) → Sells(West, x, Nono)
  – Missile(x) ⇒ Weapon(x)
  – Enemy(x, America) → Hostile(x)
  – American(West)
  – Enemy(Nono, America)

• Converted to CNF
  – ¬American(x) ∨ ¬Weapon(y) ∨ ¬Sells(x, y, z) ∨ ¬Hostile(z) ∨ Criminal(x)
  – Missile(M₁) and Owns(Nono, M₁)
  – ¬Missile(x) ∨ ¬Owns(Nono, x) ∨ Sells(West, x, Nono)
  – ¬Missile(x) ∨ Weapon(x)
  – ¬Enemy(x, America) ∨ Hostile(x)
  – American(West)
  – Enemy(Nono, America)

• Query: ¬Criminal(West)