Inference in First-Order Logic

Philipp Koehn

16 March 2017
# A Brief History of Reasoning

<table>
<thead>
<tr>
<th>Year</th>
<th>Person</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>450 B.C.</td>
<td>Stoics</td>
<td>propositional logic, inference (maybe)</td>
</tr>
<tr>
<td>322 B.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:

\[ \forall v \alpha \overset{\text{SUBST} \left( \{v/g\}, \alpha \right)}{\implies} \]

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

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

\[
\begin{align*}
King(John) \land Greedy(John) & \implies Evil(John) \\
King(Richard) \land Greedy(Richard) & \implies Evil(Richard) \\
King(\text{Father}(John)) \land Greedy(\text{Father}(John)) & \implies Evil(\text{Father}(John)) \\
\vdots 
\end{align*}
\]
Existential Instantiation

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

\[
\exists v \; \alpha \\
\text{SUBST}(\{v/k\}, \alpha)
\]

- E.g., \( \exists x \; \text{Crown}(x) \land \text{OnHead}(x, \text{John}) \) yields

\[
\text{Crown}(C_1) \land \text{OnHead}(C_1, \text{John})
\]

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:
  \[
  \forall x \ King(x) \land Greedy(x) \implies Evil(x)
  \]
  \[
  King(John)
  \]
  \[
  Greedy(John)
  \]
  \[
  Brother(Richard, John)
  \]

- Instantiating the universal sentence in all possible ways, we have
  \[
  King(John) \land Greedy(John) \implies Evill(John)
  \]
  \[
  King(Richard) \land Greedy(Richard) \implies Evil(Richard)
  \]
  \[
  King(John)
  \]
  \[
  Greedy(John)
  \]
  \[
  Brother(Richard, John)
  \]

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

- Claim: a ground sentence* 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., $\text{Father}\left(\text{Father}\left(\text{Father}(\text{John})\right)\right)$

- 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
  \[ \forall x \ King(x) \land Greedy(x) \rightarrow Evil(x) \]
  \[ King(John) \]
  \[ \forall y \ Greedy(y) \]
  \[ Brother(Richard, John) \]

  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 \( p \cdot n^k \) 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

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

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

- **\text{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><code>Knows(John, x)</code></td>
<td><code>Knows(John, Jane)</code></td>
<td><code>{x/Jane}</code></td>
</tr>
<tr>
<td><code>Knows(John, x)</code></td>
<td><code>Knows(y, Mary)</code></td>
<td><code>{x/Mary, y/John}</code></td>
</tr>
<tr>
<td><code>Knows(John, x)</code></td>
<td><code>Knows(y, Mother(y))</code></td>
<td></td>
</tr>
<tr>
<td><code>Knows(John, x)</code></td>
<td><code>Knows(x, Mary)</code></td>
<td></td>
</tr>
</tbody>
</table>
Unification

- **UNIFY(α, β) = θ if αθ = βθ**

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>θ</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

- **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>{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>fail</td>
</tr>
</tbody>
</table>

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

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

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

- All variables assumed universally quantified

\[ p_1', p_2', \ldots, p_n', \ (p_1 \land p_2 \land \ldots \land p_n \Rightarrow q) \]

\[ q^\theta \] where \( p_i'\theta = p_i\theta \) for all \( i \)

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

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

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

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

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

\[ \implies \text{Result of modus ponens: } q^\theta \text{ is } \text{Evil}(\text{John}) \]
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:
  \[ \text{American}(x) \land \text{Weapon}(y) \land \text{Sells}(x,y,z) \land \text{Hostile}(z) \implies \text{Criminal}(x) \]

- Nonon ... has some missiles, i.e., \( \exists x \ \text{Owns}(\text{Nono},x) \land \text{Missile}(x) \):
  \( \text{Owns}(\text{Nono},M_1) \) 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 Nonon, an enemy of America ...
  \( \text{Enemy}(\text{Nono},\text{America}) \)
Forward Chaining Proof

\begin{align*}
\text{American(West)} & \quad \text{Missile(M1)} & \quad \text{Owns(Nono,M1)} & \quad \text{Enemy(Nono,America)}
\end{align*}
Forward Chaining Proof

(Note: $\forall x \ Missile(x) \land Owns(Nono,x) \implies Sells(West,x,Nono)$)
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 KB)
  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 $k-1$.
  - 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**

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

- **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

\begin{algorithm}
\textbf{function} FOL-FC-Ask(\textit{KB}, \textit{\alpha}) \textbf{returns} a substitution or \textit{false} \\
\textbf{repeat until} \textit{new} is empty \\
\hspace{1em} \textit{new} \leftarrow \emptyset \\
\hspace{1em} \textbf{for each} sentence \textit{r} in \textit{KB} do \\
\hspace{2em} \textit{(} p_1 \land \ldots \land p_n \land\land q \textit{)} \leftarrow \text{STANDARDIZE-APART}(\textit{r}) \\
\hspace{2em} \textbf{for each} \textit{\theta} such that \textit{(} p_1 \land \ldots \land p_n \textit{)}\textit{\theta} = \textit{(} p'_1 \land \ldots \land p'_n \textit{)}\textit{\theta} \\
\hspace{3em} \text{for some} \textit{p'_1}, \ldots, \textit{p'_n} in \textit{KB} \\
\hspace{2em} q' \leftarrow \text{SUBST}(\textit{\theta}, q) \\
\hspace{2em} \textbf{if} \textit{q'} is not a renaming of a sentence already in \textit{KB} or \textit{new} then do \\
\hspace{3em} \text{add} \textit{q'} \text{ to} \textit{new} \\
\hspace{3em} \phi \leftarrow \text{UNIFY}(\textit{q'}, \textit{\alpha}) \\
\hspace{3em} \textbf{if} \phi \text{ is not} \textit{fail} then \textbf{return} \phi \\
\hspace{1em} \text{add} \textit{new} \text{ to} \textit{KB} \\
\textbf{return} \textit{false}
\end{algorithm}
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
Backward Chaining Example

\[
\text{Criminal}(\text{West})
\]

\(\{x/\text{West}\}\)

- \(\text{American}(x)\)
- \(\text{Weapon}(y)\)
- \(\text{Sells}(x,y,z)\)
- \(\text{Hostile}(z)\)
Backward Chaining Example
Backward Chaining Example
Backward Chaining Example

\[ \text{Criminal}(\text{West}) \]

\[ \{ x/\text{West}, y/M1 \} \]

- \[ \text{American}(\text{West}) \]
  - \[ \{ \} \]
- \[ \text{Weapon}(y) \]
- \[ \text{Sells}(x,y,z) \]
- \[ \text{Hostile}(z) \]
- \[ \text{Missile}(y) \]
  - \[ \{ y/M1 \} \]
Backward Chaining Example

- **Criminal(West)**
  - **American(West)**
    - {}
  - **Weapon(y)**
  - **Sells(West, M1, z)**
    - {}
  - **Hostile(z)**
  - **Missile(y)**
    - {}
  - **Missile(M1)**
  - **Owns(Nono, M1)**
Backward Chaining Example

Diagram showing the backward chaining process with nodes labeled as follows:

- Criminal(West)
- American(West)
- Weapon(y)
- Sells(West,M1,z)
- Hostile(Nono)
- Missile(y)
- Missile(M1)
- Owns(Nono,M1)
- 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 (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
    \(\text{new_goals} \leftarrow [p_1, \ldots, p_n|\text{REST}(goals)]\)
    \(answers \leftarrow \text{FOL-BC-Ask}(KB, new_goals, \text{COMPOSE}(\theta', \theta)) \cup answers\)
return \(answers\)
logic programming
Logic Programming

- Sound bite: computation as inference on logical KBs

<table>
<thead>
<tr>
<th>Logic programming</th>
<th>Ordinary programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Identify problem</td>
<td>Identify problem</td>
</tr>
<tr>
<td>2. Assemble information</td>
<td>Assemble information</td>
</tr>
<tr>
<td>3. Tea break</td>
<td>Figure out solution</td>
</tr>
<tr>
<td>4. Encode information in KB</td>
<td>Program solution</td>
</tr>
<tr>
<td>5. Encode problem instance as facts</td>
<td>Encode problem instance as data</td>
</tr>
<tr>
<td>6. Ask queries</td>
<td>Apply program to data</td>
</tr>
<tr>
<td>7. Find false facts</td>
<td>Debug procedural errors</td>
</tr>
</tbody>
</table>

- 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\(_n\).

```prolog
criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
missile(M1).
owns(Nono,M1).
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 \( \text{alive}(X) :- \text{not} \ \text{dead}(X). \)
  \( \text{alive}(joe) \) succeeds if \( \text{dead}(joe) \) fails
resolution
Resolution: Brief Summary

• Full first-order version:

\[
\ell_1 \lor \ldots \lor \ell_k, \quad m_1 \lor \ldots \lor m_n
\]

\[
(\ell_1 \lor \ldots \lor \ell_{i-1} \lor \ell_{i+1} \lor \ldots \lor \ell_k \lor m_1 \lor \ldots \lor m_{j-1} \lor m_{j+1} \lor \ldots \lor m_n)\theta
\]

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

• For example,

\[
\neg \text{Rich}(x) \lor \text{Unhappy}(x)
\]

\[
\text{Rich}(\text{Ken})
\]

\[
\frac{}{\text{Unhappy}(\text{Ken})}
\]

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:
\[ \forall x \ [\forall y \ \text{Animal}(y) \implies \text{Loves}(x, y)] \implies [\exists y \ \text{Loves}(y, x)] \]

1. Eliminate biconditionals and implications

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

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

\[ \forall x \ [\exists y \ \neg (\neg \text{Animal}(y) \lor \text{Loves}(x, y))] \lor [\exists y \ \text{Loves}(y, x)] \]
\[ \forall x \ [\exists y \ \neg \neg \text{Animal}(y) \land \neg \text{Loves}(x, y)] \lor [\exists y \ \text{Loves}(y, x)] \]
\[ \forall x \ [\exists y \ \text{Animal}(y) \land \neg \text{Loves}(x, y)] \lor [\exists y \ \text{Loves}(y, x)] \]
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
  - \( \text{American}(x) \land \text{Weapon}(y) \land \text{Sells}(x,y,z) \land \text{Hostile}(z) \implies \text{Criminal}(x) \)
  - \( \text{Missile}(M_1) \text{ and } \text{Owns}(\text{Nono}, M_1) \)
  - \( \forall x \ \text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \implies \text{Sells}(\text{West}, x, \text{Nono}) \)
  - \( \text{Missile}(x) \implies \text{Weapon}(x) \)
  - \( \text{Enemy}(x, \text{America}) \implies \text{Hostile}(x) \)
  - \( \text{American}(\text{West}) \)
  - \( \text{Enemy}(\text{Nono, America}) \)

- Converted to CNF
  - \( \neg \text{American}(x) \lor \neg \text{Weapon}(y) \lor \neg \text{Sells}(x,y,z) \lor \neg \text{Hostile}(z) \lor \text{Criminal}(x) \)
  - \( \text{Missile}(M_1) \text{ and } \text{Owns}(\text{Nono}, M_1) \)
  - \( \neg \text{Missile}(x) \lor \neg \text{Owns}(\text{Nono}, x) \lor \text{Sells}(\text{West}, x, \text{Nono}) \)
  - \( \neg \text{Missile}(x) \lor \text{Weapon}(x) \)
  - \( \neg \text{Enemy}(x, \text{America}) \lor \text{Hostile}(x) \)
  - \( \text{American}(\text{West}) \)
  - \( \text{Enemy}(\text{Nono, America}) \)

- Query: \( \neg \text{Criminal}(\text{West}) \)
Resolution Proof

\[ \neg \text{American}(x) \lor \neg \text{Weapon}(y) \lor \neg \text{Sells}(x,y,z) \lor \neg \text{Hostile}(z) \lor \text{Criminal}(x) \]

\[ \neg \text{Criminal}(\text{West}) \]

\[ \text{American}(\text{West}) \]

\[ \neg \text{Missile}(x) \lor \text{Weapon}(x) \]

\[ \neg \text{Weapon}(y) \lor \neg \text{Sells}(\text{West},y,z) \lor \neg \text{Hostile}(z) \]

\[ \text{Missile}(\text{M1}) \]

\[ \neg \text{Missile}(y) \lor \neg \text{Sells}(\text{West},y,z) \lor \neg \text{Hostile}(z) \]

\[ \neg \text{Missile}(x) \lor \neg \text{Owns}(\text{Nono},x) \lor \text{Sells}(\text{West},x,\text{Nono}) \]

\[ \neg \text{Sells}(\text{West},\text{M1},z) \lor \neg \text{Hostile}(z) \]

\[ \text{Missile}(\text{M1}) \]

\[ \neg \text{Missile}(\text{M1}) \lor \neg \text{Owns}(\text{Nono},\text{M1}) \lor \neg \text{Hostile}(\text{Nono}) \]

\[ \text{Owns}(\text{Nono},\text{M1}) \]

\[ \neg \text{Owns}(\text{Nono},\text{M1}) \lor \neg \text{Hostile}(\text{Nono}) \]

\[ \neg \text{Enemy}(x,\text{America}) \lor \text{Hostile}(x) \]

\[ \neg \text{Hostile}(\text{Nono}) \]

\[ \text{Enemy}(\text{Nono},\text{America}) \]

\[ \text{Enemy}(\text{Nono},\text{America}) \]