First-Order Logic

Rather powerful representation and reasoning system.

Very well understood and extensively studied (a couple thousand years!).

Many fancy knowledge representation formalisms—semantic nets, frames, scripts—are basically sugar-coated variants of (part of) it.
Components of First-Order Logic

Variables, constant symbols, and function symbols are used to build terms:
- $x$, Bill, $6$, FatherOf($x$), Height(FatherOf(Bill)), Log($3 + y$), ...

Relational symbols are applied to terms to build predicates:
- Even($x$), Married(Bill, Hillary), Loves($x$, MotherOf($x$)), ...

Predicates and logical constants are used to build sentences:
- $\exists x \neg$Loves($x$, MotherOf($x$)), $\forall x$ Even($x$) $\rightarrow$ Odd($x + 1$), ...

Variables, constant symbols, and function symbols are used to build terms:
- For all (universal quantifier), There exists (existential quantifier), Equal, Logical operators: As in Propositional Logic plus
  - Relational symbols (with arities)
  - Functional symbols (with arities)

Language of FOL: Symbols
**Language of FOL: Terms**

- A variable is a term.
- A constant symbol is a term.
- If $f$ is an $n$-ary function symbol and $t_1, \ldots, t_n$ are $n$ terms, $f(t_1, \ldots, t_n)$ is a term.
- Nothing else is a term.

**Language of FOL: Sentences**

- $true$ or $false$ is a sentence.
- If $t_1, t_2$ are terms, $t_1 = t_2$ is a sentence.
- If $p$ is an $n$-ary relation symbol and $t_1, \ldots, t_n$ are $n$ terms, $p(t_1, \ldots, t_n)$ is a sentence.
- If $\phi$ is a sentence, $\neg \phi$, $\exists x \phi$, $\forall x \phi$ are sentences.
- If $\phi_1, \phi_2$ are sentences, $\phi_1 \land \phi_2$, $\phi_1 \lor \phi_2$, $\phi_1 \rightarrow \phi_2$, $\phi_1 \iff \phi_2$ are sentences.
- Nothing else is a sentence.
Universal quantification

$\forall x \text{At}(x, \text{UIowa}) \rightarrow \text{Smart}(x)$

Common mistake: using $\land$ as the main connective with $\forall$

Typically, $\land$ is the main connective with $\forall$

A common mistake to avoid

\[ \forall x \text{At}(x, \text{UIowa}) \land \text{Smart}(x) \]

*Typically*, $\land$ is the main connective with $\forall$

At(John, UIowa) \land Smart(John)

At(KingJohn, UIowa) \land Smart(KingJohn)

At(Richard, UIowa) \land Smart(Richard)

At(KingJohn, UIowa) \land Smart(KingJohn)

\[ \ldots \land \]

*Typically*, $\land$ is the main connective with $\forall$

\[ \forall x \text{At}(x, \text{UIowa}) \land \text{Smart}(x) \]

\[ \forall x \text{At}(x, \text{UIowa}) \rightarrow \text{Smart}(x) \]

Universal quantification

Everyone at UIowa and everyone is smart:

\[ \forall x \text{At}(x, \text{UIowa}) \land \text{Smart}(x) \]

A common mistake: using $\land$ as the main connective with $\forall$

Typically, $\rightarrow$ is the main connective with $\forall$

Typically, $\land$ is the main connective with $\forall$

A common mistake to avoid

Typically, $\land$ is the main connective with $\forall$

Typically, $\land$ is the main connective with $\forall$

Typically, $\land$ is the main connective with $\forall$

Typically, $\land$ is the main connective with $\forall$

Typically, $\land$ is the main connective with $\forall$

Typically, $\land$ is the main connective with $\forall$

Typically, $\land$ is the main connective with $\forall$

Typically, $\land$ is the main connective with $\forall$

Typically, $\land$ is the main connective with $\forall$

Typically, $\land$ is the main connective with $\forall$

Typically, $\land$ is the main connective with $\forall$

Typically, $\land$ is the main connective with $\forall$

Typically, $\land$ is the main connective with $\forall$

Typically, $\land$ is the main connective with $\forall$

Typically, $\land$ is the main connective with $\forall$

Typically, $\land$ is the main connective with $\forall$

Typically, $\land$ is the main connective with $\forall$

Typically, $\land$ is the main connective with $\forall$

Typically, $\land$ is the main connective with $\forall$

Typically, $\land$ is the main connective with $\forall$

Typically, $\land$ is the main connective with $\forall$

Typically, $\land$ is the main connective with $\forall$

Typically, $\land$ is the main connective with $\forall$

Typically, $\land$ is the main connective with $\forall$

Typically, $\land$ is the main connective with $\forall$

Typically, $\land$ is the main connective with $\forall$

Typically, $\land$ is the main connective with $\forall$
Existential quantification

\[ \exists x \text{At}(x, \text{Stanford}) \land \text{Smart}(x) \]

Common mistake: using \( \land \) as the main connective with \( \exists \)

Typically, \( \exists \) is the main connective with \( \land \)

Another common mistake to avoid

Artificial Intelligence – p.9/81

Typically, \( \land \) is the main connective with \( \exists \)

Common mistake: using \( \rightarrow \) as the main connective with \( \exists \):

\[ \exists x \text{At}(x, \text{Stanford}) \rightarrow \text{Smart}(x) \]

is true if there is anyone who is not at Stanford!

Artificial Intelligence – p.10/81

Roughly speaking, equivalent to the disjunction of

\[ \forall x (\text{At}(x, \text{Stanford}) \lor \text{Smart}(x)) \land \forall x (\text{At}(x, \text{Richard}) \lor \text{Smart}(x)) \land \forall x (\text{At}(x, \text{Stanford}) \lor \text{Smart}(x)) \]

\[ \exists x (\text{At}(x, \text{Stanford}) \lor \text{Smart}(x)) \land \exists x (\text{At}(x, \text{Richard}) \lor \text{Smart}(x)) \land \exists x (\text{At}(x, \text{Stanford}) \lor \text{Smart}(x)) \land \exists x (\text{At}(x, \text{Stanford}) \lor \text{Smart}(x)) \land \exists x (\text{At}(x, \text{Stanford}) \lor \text{Smart}(x)) \]

\[ \exists x (\text{At}(x, \text{Stanford}) \lor \text{Smart}(x)) \]

is true in an interpretation \( I \) if \( p \) is true with \( x \) being

\[ p \land \exists x (\text{At}(x, \text{Stanford}) \lor \text{Smart}(x)) \]

Some at Stanford is smart:

\[ \forall y \text{At}(y, \text{Stanford}) \land \text{Smart}(y) \]

\[ \forall y \text{At}(y, \text{Stanford}) \land \forall y \text{Smart}(y) \]

\[ \forall y \text{At}(y, \text{Stanford}) \land \forall y \text{Smart}(y) \]

Existentiál quantification
Properties of quantifiers

**∀** x **∀** y is the same as **∀** y **∀** x (**why??**)

**∃** x **∃** y is the same as **∃** y **∃** x (**why??**)

**∃** x **∀** y is not the same as **∀** y **∃** x

**∃** x **∀** y Loves (x, y)

"There is a person who loves everyone in the world"

**∀** y **∃** x Loves (x, y)

"Everyone in the world is loved by at least one person"

**Quantifier duality**: each can be expressed using the other

"Brothers are siblings"

∀ x, y Brother (x, y) → Sibling (x, y).

"Sibling" is symmetric

∀ y Loves x → Loves x, y

"There is a person who loves everyone in the world"

∃ x Loves x, y → Loves y, x

"Everyone in the world is loved by at least one person"

Artificial Intelligence – p.12/81

Fun with sentences
Fun with sentences

Brothers are siblings

∀x, y Brother(x, y) → Sibling(x, y).

"Sibling" is symmetric

∀x, y Sibling(x, y) ⇔ Sibling(y, x).

One's mother is one's female parent

∀x, y Mother(x, y) ⇔ (Female(x) ∧ Parent(x, y)).

A first cousin is a child of a parent's sibling

∀x, ps, pt Parent(ps, pt) ∧ Sibling(x, ps) ⇔ (∃p, ps Parent(p, ps) ∧ Parent(p, pt) ∧ Sibling(x, p)).

Brothers are siblings

∀x, y Brother(x, y) → Sibling(x, y).

"Sibling" is symmetric

∀x, y Sibling(x, y) ⇔ Sibling(y, x).

One's mother is one's female parent

∀x, y Mother(x, y) ⇔ (Female(x) ∧ Parent(x, y)).
When in doubt, use parentheses:

\[((x)_S \leftarrow (((x)_R \lor (x)_Q) \land ((x)_D \leftarrow))) \land (A \lor \neg(x)_S)\]  

With this ordering, (1) actually is:

\[\{(E, \lambda{i}, \neg, \lor, \land), \Rightarrow, \leftarrow\}\]  

As defined, the syntax of FOL is ambiguous.

A Note on the Syntax of FOL

\[
\cdots | < | Apple | RedBrother | \cdots | D | P | \cdots | + | Cosine | Height | FatherOf | \cdots | \forall | Variable | \forall | Variable | Variable | \forall | \land | \lor | \rightarrow | \neg | \exists | \Rightarrow | \leftarrow | \\
Variable | Variable | Variable | Sentence | Sentence | Complex | Complex | Complex | Complex | Complex | Complex | Complex | Complex |
\]
Semantics of First-Order Logic

Interprets

Constant Symbols and Variables

Denote (stand for) objects/individuals.

Example:

Dog143 is a constant symbol denoting a particular dog.

peach is a variable ranging over peaches.

Under another interpretation Dog143 could denote another dog, or maybe something else altogether.

Similarly, peach might as well range over apples.

= as equality (i.e., the identity relation).

functional symbols as relations over objects;
relational symbols as functions from objects to objects;
Semantics of First-Order Logic

Interprets

the universal quantifier (essentially) as an infinite

\( \forall x \) \( \text{Red}(x) \equiv (x) \text{Red}(1) \text{Red}(2) \text{Red}(3) \text{Red}(4) x \) 

the existential quantifier (essentially) as an infinite

\( \exists x \) \( \text{Red}(x) \equiv (x) \text{Red}(1) \text{Red}(2) \text{Red}(3) \text{Red}(4) x \) 

true, false, \( \& \), \( \lor \), \( \neg \), \( \leftrightarrow \) as in propositional logic:

\( \cdots \land (\forall x) \text{Red}(x) \land (\forall x) \text{Red}(1) \text{Red}(2) \text{Red}(3) \text{Red}(4) x \) 

\( \cdots \lor (\forall x) \text{Red}(x) \lor (\forall x) \text{Red}(1) \text{Red}(2) \text{Red}(3) \text{Red}(4) x \) 

Semantics of FOL: Examples

<table>
<thead>
<tr>
<th>Sentence</th>
<th>Intuitive meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ( \forall x ) ( \text{Apple}(x) \lor (\forall x) \text{Pear}(x) )</td>
<td>Object A is above object B</td>
</tr>
<tr>
<td>2. ( \forall x ) ( \text{Apple}(x) ) ( \rightarrow ) ( \text{Red}(x) )</td>
<td>Object A is above object B</td>
</tr>
<tr>
<td>3. ( \forall x ) ( \text{Apple}(x) ) ( \land ) ( (\forall x) \text{Red}(x) )</td>
<td>Object A is above object B</td>
</tr>
<tr>
<td>4. ( \forall x ) ( \text{Apple}(x) ) ( \lor ) ( (\forall x) \text{Red}(x) )</td>
<td>Object A is above object B</td>
</tr>
<tr>
<td>5. ( \forall x ) ( \text{Apple}(x) ) ( \rightarrow ) ( \exists y ) ( y &gt; x )</td>
<td>Object A is above object B</td>
</tr>
<tr>
<td>6. ( \forall x ) ( \exists y ) ( y &gt; x )</td>
<td>Object A is above object B</td>
</tr>
<tr>
<td>7. ( \forall x ) ( \text{Red}(x) ) ( \land ) ( (\forall x) \text{Red}(x) )</td>
<td>Object A is above object B</td>
</tr>
<tr>
<td>8. ( \forall x ) ( \exists y ) ( \text{Loves}(x, y) )</td>
<td>Object A is above object B</td>
</tr>
<tr>
<td>9. ( \exists x ) ( \forall y ) ( \text{Loves}(x, y) )</td>
<td>Object A is above object B</td>
</tr>
<tr>
<td>10. ( \exists y ) ( \forall x ) ( \text{Loves}(x, y) )</td>
<td>Object A is above object B</td>
</tr>
</tbody>
</table>

Artificial Intelligence – p.19/81
Semantics of FOL: Examples

Sentence Intuitive meaning

10. \(\forall x (\text{Bird}(x) \land \neg \text{Ostrich}(x)) \rightarrow \text{F lies}(x)\)

11. \(\forall x (\text{Fish}(x) \land \neg \text{Ostrich}(x)) \rightarrow \text{F lies}(x)\)

12. \(\forall x \text{Flies}(x) \iff (\text{Bird}(x) \lor \text{P lane}(x))\)

13. \(\forall x \text{Person}(x) \rightarrow (\text{W}(x) \lor \text{M}(x)) \land \neg (\text{W}(x) \land \text{M}(x))\)

14. \(\forall x \text{Age}(x) < \text{Age}(\text{FatherOf}(x))\)

Note:

A sentence is open if it contains free variables; it is closed if all its free variables are bound.

Free and Bound Variables

An occurrence of variable in a sentence is free if it is not in the scope of any quantifier with the same variable. A non-free variable occurrence is bound.

A sentence is open if it contains free variables; it is closed otherwise.

Artificial Intelligence – p.21/81
Formalizing Knowledge in FOL

First Step: Fix a Universe of Discourse, that is, the set of objects of interest.

Relations

Notes: The universe may be infinite, even uncountable!

Here the universe is {block a, block b, ...}

Example: The Blocks World

Artificial Intelligence – p.23/81

Artificial Intelligence – p.24/81

Note: Less Than over the naturals as:

Clear := \{⟨a⟩, ⟨d⟩\}

Read := as is defined

\{⟨p⟩, ⟨q⟩\} := On

\{⟨r, e⟩, ⟨e, p⟩, ⟨r, e⟩, ⟨e, q⟩, ⟨q, a⟩\} := On

\{⟨e, t⟩\} =: \{(0, 1), (0, 2), ..., (1, 2), (1, 3), ...\}

Example: Some relations in the Blocks World

\{(0, 1), (1, 2), ..., (1, 3), ...

A relation of arity \(n\) is a subset of the Cartesian product \(U \times \cdots \times U\) \(n\) times.

Let \(U\) be the universe of discourse.

The integers, the reals, ...
Functions

A function in one variable is simply a functional binary relation. A relation \( f \) over \( U \times U \) is functional if for all \( x \in U \) there is one and only one \( y \in U \) such that \( \langle x, y \rangle \in f \).

Hence, instead of writing \( \langle x, y \rangle \in f \) we write \( y = f(x) \).

Functions in \( n > 1 \) variables are defined similarly.

Examples of Functions

Most mathematical functions \(+, \times, e^x, \ldots\)

Some relations in the Blocks World:

<table>
<thead>
<tr>
<th>Color</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>{\langle a, \text{blue} \rangle, \langle b, \text{red} \rangle, \langle c, \text{blue} \rangle, \ldots }</td>
<td>{\langle a, b \rangle, \langle b, c \rangle, \langle c, t \rangle, \langle d, e \rangle, \langle e, t \rangle }</td>
</tr>
</tbody>
</table>

We may define function \( \text{Color}(x) \) such that

\[ \text{Color}(a) = \text{blue}, \text{Color}(b) = \text{red}, \ldots \]

Note: Functions in FOL are always total, i.e., defined over the entire universe of discourse.

Most mathematical functions are defined similarly.

If \( f \) is a function, \( f \in \langle x, z \rangle \) implies \( (f \in \langle z, x \rangle) \lor (f \in \langle y, x \rangle) \).

A relation \( f \) over \( U \times U \) is functional if for all \( x \in U \) there is one and only one \( y \in U \) such that \( \langle x, y \rangle \in f \).

A function is one variable is simply a functional binary relation.

Functions
An Interpretation in the Blocks World

Constant Symbols:
A, B, C, D, E, T

Function Symbols:
Support

Relation Symbols:
On, Above, Clear

An interpretation is a pair $(D, \sigma)$ where

- $D$ is a set of objects (or domain),
- $\sigma$ is mapping from variables to objects in $D$,
- $\sigma$ is a relation over $D$.

The interpretation $I$ of the blocks world is:

- $\text{Support}_I = \{(a, b), (b, c), (d, e), (e, t), (t, t)\}$
- $\text{On}_I = \{(a, b), (b, c), (d, e)\}$
- $\text{Above}_I = \{(a, b), (a, c), (b, c), (d, e)\}$
- $\text{Clear}_I = \{(a), (d)\}$

The semantics of First-Order Logic:

- An interpretation is a pair $(D, \sigma)$
- $\{p\}^I = \{\text{true}\}$
- $\{(\phi \land \psi)\}^I = \text{true}$
- $\{(\phi \lor \psi)\}^I = \text{true}$
- $\{(\neg \phi)\}^I = \text{true}$
- $\{(\exists x \phi)\}^I = \text{true}$
- $\{(\forall x \phi)\}^I = \text{true}$
Semantics of First-Order Logic

Let \((D, \sigma)\) be an interpretation and \(E\) an expression of FOL. We write \([E]_{D\sigma}\) to denote the meaning of \(E\) in the domain \(D\) under the variable assignment \(\sigma\).

The meaning \([t]_{D\sigma}\) of a term \(t\) is an object of \(D\).

The meaning \([F(t_1, \ldots, t_n)]_{D\sigma}\) for function symbols \(F\) of arity \(n\) is inductively defined as follows.

1. The meaning \([x]_{D\sigma}\) is \(\sigma(x)\) for variables \(x\).
2. The meaning \([C]_{D\sigma}\) is \(C\) for constant symbols \(C\).
3. The meaning \([F(t_1, \ldots, t_n)]_{D\sigma}\) is \(F(D[t_1]_{D\sigma}, \ldots, D[t_n]_{D\sigma})\) for function symbols \(F\) of arity \(n\).

The meaning \([E]_{D\sigma}\) of an expression \(E\) is defined similarly.
The meaning of formulas built with the other logical symbols can be defined by reduction to the previous symbols.

\[
\begin{align*}
\phi_1 \land \phi_2 & \Leftrightarrow \neg (\neg \phi_1 \lor \neg \phi_2) \\
\phi_1 \rightarrow \phi_2 & \Leftrightarrow \neg \phi_1 \lor \phi_2 \\
\phi_1 \leftrightarrow \phi_2 & \Leftrightarrow (\phi_1 \rightarrow \phi_2) \land (\phi_2 \rightarrow \phi_1) \\
\forall x \phi & \Leftrightarrow \neg \exists x \neg \phi
\end{align*}
\]

The meaning of a formula \( \phi \) is defined as follows:

- If a sentence is closed (no free variables), its meaning does not depend on the domain.

- For any \( x \),

\[
\mathcal{D}[\forall x \phi] = \phi_{\mathcal{D}}[\forall x \phi(x)] = \phi_{\mathcal{D}}[\forall x \phi(x)]
\]

- If a sentence is closed (no free variables), its meaning does not depend on the variable assignment (although its meaning may depend on the domain):

\[
\mathcal{D}[\forall x \exists y R(x, y)] = \mathcal{D}[\forall x \exists y R(x, y)]'
\]

The meaning of First-Order Logic

Semantics of First-Order Logic
Models for Sentences

An interpretation \((D, \sigma)\) satisfies a sentence \(\varphi\), or is a model for \(\varphi\), if \(\llbracket \varphi \rrbracket_{D, \sigma} = \text{true}\).

A sentence is satisfiable if it has at least one model.

Examples:
\[ \forall x \ x \geq y, \ P(x) \]

A sentence is unsatisfiable if it has no models.

Examples:
\[ P(x) \land \neg P(x), \neg (x = x) \]

A sentence \(\varphi\) is valid if every interpretation is a model.

Examples:
\[ (x = x), (x)_{D}, (x)_{D} \leftarrow (x)_{D} \]

A sentence \(\varphi\) is invalid/unsatisfiable if it has no models.

Examples:
\[ \{x < x + 1 \mid x > 0, \forall x \} \]

As in Propositional Logic, \(\Gamma\) entails a sentence \(\varphi\) if every model of \(\Gamma\) is also a model of \(\varphi\).

Examples:
\[ \{ (x)_{D} \leftarrow (x)_{D} \} \]

\(\Gamma\) is unsatisfiable, or inconsistent, if it has no models.

Examples:
\[ \{ x < x + 1 \mid x > 0, \forall x \} \]

A set \(\Gamma\) of sentences is satisfiable if it has at least one model.

Examples:
\[ \{ P(x), \neg P(x) \} \]

A set \(\Gamma\) of sentences satisfies a set \(\Gamma\) of sentences, or \(\Gamma\) is a model for \(\Gamma\), if it is a model for every sentence in \(\Gamma\).

Models for Sets of Sentences

They are always true:

Valid sentences do not tell us anything about the world.

\(\varphi\) is valid/unsatisfiable iff \(\varphi\) is valid/unsatisfiable.

\[ x = x, \ (x)_{D} \leftarrow (x)_{D} \]

For \(\varphi\):

A sentence is valid if every interpretation is a model.

Examples:
\[ (x = x), (x)_{D}, (x)_{D} \leftarrow (x)_{D} \]

A sentence is unsatisfiable if it has no models.

Examples:
\[ (x)_{D}, (x)_{D} \leftarrow (x)_{D} \]

A sentence is satisfiable if it has at least one model.

Examples:
\[ \forall x \ x < x + 1 \mid x > 0, \forall x \]

An interpretation \((D, \sigma)\) satisfies a sentence \(\varphi\) if \(\varphi\) is a model for \(\varphi\).

\[ \llbracket \varphi \rrbracket_{D, \sigma} = \text{true} \]
Models for FOL: Lots!

We can enumerate the models for a given FOL sentence:

For each number of universe elements $n$

For each $k$-ary predicate $P$ in the sentence

For each possible $k$-ary relation on $n$ objects

For each constant symbol $C$ in the sentence

For each one of $n$ objects mapped to $C$

Enumerating models is not going to be easy!

Sentences can be seen as constraints on the set of all possible interpretations.

Possible Interpretations Semantics

A sentence denotes either no interpretations or an infinite number of them.

A sentence denotes either $S_1 \subseteq S_2$, $S_1 \cap S_2$, $S_1 \cup S_2$, or $S_1 \setminus S_2$.

If $\varphi_1$ denotes a set of interpretations $S_1$ and $\varphi_2$ denotes all the possible interpretations that satisfy it (the models of $\varphi$), then $\varphi_1 \lor \varphi_2$ denotes $S_1 \cup S_2$, $\varphi_1 \land \varphi_2$ denotes $S_1 \cap S_2$, $\neg \varphi_1$ denotes $S \setminus S_1$, and $\varphi_1 \models \varphi_2$ if $S_1 \subseteq S_2$.

A sentence denotes either no interpretations or an infinite number of them!
\[ \theta = (\forall w_1 \ldots \forall w_n \phi) = IV \]
and
\[ (\exists \tilde{x}_1 \ldots \exists \tilde{x}_n \psi) = I \]
where
\[ \theta(IV \land T) \]
\[ \frac{\text{UNIFY}(\ell_1, m_1)}{\ell_1 \land IV} \]
\[ \frac{\text{UNIFY}(\ell_2 \lor \cdots \lor \ell_k, m_2 \lor \cdots \lor m_n)}{\ell_2 \lor \cdots \lor \ell_k \land m_2 \lor \cdots \lor m_n} \]

Or equivalently,
\[ \theta = (\exists \tilde{x}_1 \ldots \exists \tilde{x}_n \psi) \]
\[ \frac{\text{UNIFY}(\ell_1, m_1)}{\ell_1 \land \psi} \]

where
\[ \text{UNIFY}(\ell_i, m_j) = \theta \]

Full first-order version:

\[ \text{Resolution} \]

... | < | apple | red brother | ... | \phi | \psi | ::= \text{relation symbol}
... | + | cosine | height | father of | ::= \text{function symbol}
... | \alpha | ... | \beta | 0 | ... | ::= \text{constant symbol}
... | \text{variable} | ... | \text{variable} | ::= \text{variable}
... | \text{variable} | \in | \text{variable} | ::= \text{variable}
... | \text{variable} | \land | \text{variable} | ::= \text{variable}
... | \text{variable} | \lor | \text{variable} | ::= \text{variable}
... | \text{variable} | \forall | \text{variable} | ::= \text{variable}
... | \text{variable} | \exists | \text{variable} | ::= \text{variable}
... | \text{variable} | \in | \text{variable} | ::= \text{variable}
... | \text{sentence} | ::= \text{sentence}
... | \text{sentence} | \land | \text{sentence} | ::= \text{sentence}
... | \text{sentence} | \lor | \text{sentence} | ::= \text{sentence}
... | \text{sentence} | \forall | \text{sentence} | ::= \text{sentence}
... | \text{sentence} | \exists | \text{sentence} | ::= \text{sentence}
... | \text{sentence} | \land | \text{sentence} | ::= \text{sentence}
... | \text{sentence} | \lor | \text{sentence} | ::= \text{sentence}
... | \text{sentence} | ::= \text{sentence}

Language of FOL: Grammar
Resolution Proofs by Refutation

Let \( \Gamma = \{ \neg P(w) \lor Q(w), \neg Q(y) \lor S(y), P(x) \lor R(x), \neg R(z) \lor S(z) \} \).

To prove \( S(A) \) from \( \Gamma \), try to prove false from \( \Gamma \cup \{ \neg S(A) \} \).

\[
\{ x/y \} = ((f)\forall x.Feathers(x).Fathers(x).Bird(x) \lor (f)\forall x.Fathers(x).Bird(x).Feathers(x)) \\
\ 
((f)\forall x.Bird(x).Feathers(x).Fathers(x).Bird(x)) \\
\{ y/x \} \\
\{ (z) \land (z) \land R, (x) \land (x) \land (f) \land (f) \land (f) \land (m) \land (m) \land (p) \land (p) \} = \top
\]

Examples
Unification

A variable unifies with any term in which it does not occur.

A constant symbol unifies only with itself.

Two compound terms (or two atomic sentences) unify only if their top symbol is the same, the arguments of that symbol are applied to the same number of arguments, and no variable occurs.

A constant symbol unifies only with itself.

A variable unifies with any term in which it does not occur.

Examples of Unification

- \( x \) and \( A \) unify but \( A \) and \( B \) do not.
- \( x \) and \( F(A, H(x)) \) do not unify (\( x \) occurs in \( F(A, H(x)) \)).
- \( F(H(x)) \) and \( G(H(x)) \) do not unify (different top symbols).
- \( F(x) \) and \( F(x, y) \) do not unify (different number of arguments).
- \( F(t_1, t_2, \ldots, t_n) \) and \( F(s_1, s_2, \ldots, s_n) \) unify if \( t_1 \) and \( s_1 \) unify, \( t_2 \) and \( s_2 \) unify, \ldots, \( t_n \) and \( s_n \) unify.

Artificial Intelligence – p.41/81
θ is a unifier of \( t_1 \) and \( t_2 \) if \( \theta(t_1) \) is identical to \( \theta(t_2) \).

Some terms may have more than one unifier:

\[(\forall \theta)(t_1 \theta) \text{ is identical to } (t_2 \theta), \]

\( \theta \) is a unifier of \( t_1 \) and \( t_2 \) if \( t_1 \theta \) and \( t_2 \theta \) are.
The Transformation Rules

Decompose.

\[ \{ t_i = a \} \cup \{ f(t_1, \ldots, t_n) = ? \} \rightarrow \{ t_i = a \} \cup \{ t_i = s_i \} : 1 \leq i \leq n \] 

Orient.

\[ \{ t = v \} \cup \{ f(t_1, \ldots, t_m) = ? \} \rightarrow \{ v = t \} \cup \{ \} \] 

Delete.

\[ \{ v = t \} \rightarrow \{ v = v \} \cup \{ t = t \} \] 

Mismatch.

\[ \{ v = t \} \rightarrow \{ \} \cup \{ \} \]

Eliminate.

\[ \{ v = t \} \rightarrow \{ [v/a] \} \cup \{ v = t \} \]

Occurs.

\[ \{ t_i = a \} \cup \{ [v/a] \} \rightarrow \{ t_i = a \} \cup \{ t_i \in \{ v \} \} \]

Note: \( V(U) \) and \( V(t) \) denote the set of variables in \( U \), \( t \), respectively; \( v \) denotes a variable, \( s, t \) denote terms (possibly variables); \( f, g \) denote function/constant symbols.

The Transformation Rules

\[ u \neq m \text{ distinct or } \begin{aligned} \{ (u_1 s_1 \ldots s_1 t) \} &\rightarrow \{ (u_1 t) \} \cup \{ a \} \cup \{ t = t \} \end{aligned} \]

Mismatch.

\[ \{ a \} \cup \{ t = t \} \rightarrow \{ a \} \cup \{ t = t \} \]

Delete.

\[ \{ a \} \cup \{ t = t \} \rightarrow \{ a \} \cup \{ t = t \} \]

Orient.

\[ \{ u \leq t \leq 1 \} \cup \{ t = t \} \rightarrow \{ (u_1 t_1 \ldots t_1) \} \cup \{ (u_1 t_1 \ldots t_1) \} \]

Decompose.
The Unification Procedure: Examples

Unify:

$F(A, H(y), z)$ and $F(A, x, z)$

$\{ F(A, H(y), z) = ? F(A, x, z) \}$

Decompose $\rightarrow$

$\{ A = ? A, H(y) = ? x, z = ? z \}$

Decompose $\rightarrow$

$\{ H(y) = ? x \}$

Orient $\rightarrow$

$\{ x = ? H(y) \}$

Delete $\rightarrow$

$\{ x = ? H(y) \}$

$mgu = \{ x/H (y) \}$

Artificial Intelligence – p.47/81

The Unification Procedure: Examples

$\{(\bar{a})H/x\} = \text{mgu}$

$\{(\bar{a})H \_\_ = x\}$

$\{ x = ? H(y) \}$

$mgu = \{ x/H (y) \}$

Artificial Intelligence – p.48/81
Prove that Col. West is a criminal.

The law says that it is a crime for an American, who is 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.

Example Knowledge Base

The Unification Procedure: Examples

Unify:

\[ F(H(y), z) \text{ and } F(G(x), z) \]

\[
\begin{align*}
F(H(y), z) &= F(G(x), z) \\
\text{Decompose} &\rightarrow \\
H(y) &= G(x), z = z
\end{align*}
\]

Mismatch ➔ FAIL do not unify

Unify:

\[ F(y) \text{ and } F(H(x, y)) \]

\[
\begin{align*}
F(y) &= F(H(x, y)) \\
\text{Decompose} &\rightarrow \\
\text{Occurs} &\rightarrow \\
y &= H(x, y)
\end{align*}
\]

Occurs ➔ FAIL do not unify

The Unification Procedure: Examples
Example Knowledge Base contd.

Nono . . . has some missiles

American(x) ∨ Weapon(y) ∨ Sells(x, y) ∨ Hostile(z) ≤⇒ Criminal(x)

. . . It is a crime for an American to sell weapons to hostile nations.

Example Knowledge Base contd.
it is a crime for an American to sell weapons to hostile nations:

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

Nono has some missiles, i.e.,

\[ \exists x \text{ Owns}(\text{Nono}, x) \land \text{Missile}(x) \land \text{American}(x) \land \text{Weapon}(y) \land \text{Sells}(x, y, z) \land \text{Hostile}(z) \]

\[ \Rightarrow \text{Criminal}(z) \]

... all of its missiles were sold to it by Colonel West

\[ \forall x \text{ Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono}) \]

Missiles are weapons:

\[ \forall x \text{ Missile}(x) \Rightarrow \text{Weapon}(x) \]

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

Example Knowledge Base contd.
Example Knowledge Base contd.

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

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

- ... an enemy of America counts as "hostile":
  \[ \text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x) \]

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

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

- All of its missiles were sold to it by Colonel West
  \[ \forall x \left( \text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono}) \right) \]

- ARTIFICIAL INTELLIGENCE – p.55/81

Example Knowledge Base contd.
Why Resolution by Refutation Works

Let \( \Gamma \) be a set of clauses and \( \alpha \) an atomic sentence. To show that \( \alpha \) follows from \( \Gamma \), prove false from \( \Gamma \cup \{\neg \alpha\} \).

This proof method is sound because:

- False provable from \( \Gamma \cup \{\neg \alpha\} \) by resolution implies \( \text{false} \equiv \{\neg \alpha\} \cap \text{false} \) implies \( \Gamma \cup \{\neg \alpha\} \) is unsatisfiable implies \( \neg(\Gamma \land \neg \alpha) \) is valid implies \( \neg \Gamma \lor \alpha \) is valid implies \( \Gamma \rightarrow \alpha \) is valid.

Resolution Proof: Definite Clauses
Resolution by Refutation

1. Obtain CNF

2. Apply resolution steps to the CNF

3. If the empty clause is generated, then $\alpha \vdash \gamma$.

A proof by contradiction is also called a refutation.

Theorem: The resolution rule with factoring is a refutation-complete inference system:

If a set of clauses is unsatisfiable, then a proof by contradiction is also called a refutation.

1. Obtain $\text{CNF}(\alpha \lor \neg \beta)$

2. Apply resolution steps to the CNF

3. If the empty clause is generated, then $\alpha \vdash \neg \beta$.
Let \( \phi \) be a CNF sentence of the form below where \( \alpha_1 \lor \alpha_2 \) are both positive (or negative) literals:

\[
\alpha_1 \lor \alpha_2 \lor \beta
\]

If \( \phi_1, \phi_2 \) unify with mgu \( \theta \), then \( \theta(\alpha_1 \lor \beta) \) is a factor of \( \phi \).

Example: \( P(x) \) is a factor of \( P(x) \lor P(y) \) because \( P(x), P(y) \) unify with mgu \( \{ y/x \} \).

### Too Many Choices!

A resolution-based inference system has just one rule to apply to build a proof. However, at any step there may be several possible ways to apply the resolution rule.

Several resolution strategies have been developed in order to reduce the search space.

### Factors

Let \( \phi \) be a CNF sentence of the form below where \( \alpha_1 \lor \alpha_2 \) are both positive (or negative) literals.

\[
\{ x/\theta \} = \neg \theta \lor (\theta) \lor \phi \lor (\theta) \lor \phi
\]

Example: \( \phi \lor (x) \lor \phi \) is a factor of \( \phi \lor (x) \lor \phi \) because

\[
\theta(\phi) \lor (x) \lor \phi
\]
## Some Resolution Strategies

**Unit resolution**
- Only one clause must be used.

**Input resolution**
- One of the two clauses must be an input clause.

**Set of support**
- One of the two clauses must be from the set of support.
  - New resolvent are added into the set.

**Linear resolution**
- The latest resolvent must be used in the current resolution.

**Input resolution**
- One of the two clauses must be an input clause.

**Unit resolution**
- Unit resolution only.

---

### 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**

1. Identity problem
2. Assemble information
3. Tea break
4. Program solution
5. Encode problem instance as facts
6. Apply program to data
7. Find false facts

---

### Note

The first 3 strategies above are incomplete. The Unit resolution strategy is equivalent to the Input resolution strategy: a proof in one strategy can be converted into a proof in the other strategy.
A pure Prolog program consists of a set of definite clauses (only one positive literal). Instead of writing a clause as \( A \lor \neg B_1 \lor \neg B_2 \lor \cdots \lor \neg B_n \), we write it as \( A :- B_1, B_2, \ldots, B_n \).

A query is a negative clause (all literals are negative):

\[ ?- C_1, C_2, \ldots, C_m \]

which is the clause \( \neg C_1 \lor \neg C_2 \lor \cdots \lor \neg C_m \).

Horn Clauses = Definite Clauses + Negative Clauses

Prolog: Logic Programming Language
Prolog: A Small Example

1) father(bob, ken).
2) father(ken, joe).
3) father(joe, don).
4) mother(jan, don).

5) parent(X, Y) :- father(X, Y).
6) parent(X, Y) :- mother(X, Y).
7) ancestor(X, Y) :- parent(X, Z), parent(Z, Y).
8) ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).
9) ancestor(X, Y) :- ancestor(X, Z), parent(Z, Y).

?- ancestor(A, don).
A = ken.
A = bob.

Res6olution Proof in Prolog

It is a crime to sell weapons to hostile nations:

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

Nono has some missiles:
sells(nono, m1). missile(m1).

All of Nonos missiles were sold to it by Colonel West:
sells(west, m1). missile(m1).

Colonel West owns some missiles:
owns(west, m1). missile(m1).

Nono is an enemy of America:
hostile(nono). enemy(nono, america).

America is an enemy of all of its enemies:
enemy(A, america) :- enemy(A, Z), hostile(Z).

Colonel West is American:
american(west).

? - criminal(Who).
Who = west.

Prolog: A Small Example
Prolog examples

Depth-first search from a start state

X:
goal(e).
successor(a,b).
successor(a,c).
successor(b,c).
successor(b,d).
successor(c,d).
successor(c,e).
successor(e,b).
successor(a,p).

dfs(X) :- goal(X).
dfs(X) :- write(X), successor(X,S), dfs(S).

?- dfs(a).
a b c d e.

No need to loop over S: successor succeeds for each

a b c d e.
?- dfs(a).
dfs(X) :- write(X), successor(X,S), dfs(S).
dfs(X) :- goal(X).
successor(c,e).
successor(c,d).
successor(b,d).
successor(a,p).
successor(a,e).
goal(e).

Appending two lists to produce a third:

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

?- append([1,2], [3,4], C).
C=[1,2,3,4].
?- append([1,2], Y, [1,2,3,4]).
C=[1,2,3,4].
append([X|Z], [X|Y], [X|Z]).

Prolog examples

Depth-first search from a start state:
An Abstract Interpreter

Input: Goal $G$, list of literals, and $P$, list of definite clauses.
Output: A substitution $S$ which makes $G$ a logical consequence of $P$, or $no$ if no such instances exist.

Algorithm:

Algorithm:

```
Input: Goal $G$, list of literals, and $P$, list of definite clauses.
Output: A substitution $S$ which makes $G$ a logical consequence of $P$, or $no$ if no such instances exist.

Algorithm:
```
Traversing Search Tree

Traversing Search Tree

When writing Prolog programs, make sure a successful path in the search tree appears before any infinite search tree path.

Changes in rule order permutes the branches of the search tree. The order of clauses and the order of literals in a clause decide the structure of the search tree. Different goal orders result in search trees with different structures.

Consider the following example: We may not find a proof even if one exists, because depth-first Prolog traverses a search tree in a depth-first manner.

The results for query \( p(X), q(X) \) and query \( q(X), p(X) \) are quite different.
Unification in Prolog

A query `?- test.` will succeed for the following program:

```
less(X, succ(X)).
test :- less(Y, Y).
```

Why? This because Prolog uses an unsound unification algorithm: The occur check is omitted for the purpose of efficiency.

Another example: An infinite loop for the query `test.

test :- leq(Y, Y).
leq(X, succ(X)) :- leq(X, X).
```

Resolution Strategies in Prolog

Set of support: The query (negative clauses) is the only clause in the set of support.

Input strategy: One of the two clauses must be an input clause.

Linear strategy: Only one resolvent is saved at any time. The literals in the resolvent is organized as a stack: First in, Last Out.

The literals in one clause are treated in the order they are listed.

The literals in the other clause are treated in the order from left to right.

The first three strategies are equivalent for Horn clauses.

Why? This because Prolog uses an unsound unification algorithm: The occur check is omitted for the purpose of efficiency.

```
test :- teq(X, X).
test :- teq(X, Y).
```

Unification in Prolog
Negative Knowledge in Prolog

Pure Prolog can never deduce negative information. For example, given the program:

\[
\text{\texttt{s(a).}} \\
\text{\texttt{s(b).}} \\
\text{\texttt{t(c).}}
\]

we cannot deduce \texttt{\texttt{not s(c).}} nor \texttt{\texttt{not t(a).}}

The Closed World Assumption in Prolog

Consider again the following program P1:

\[
\text{\texttt{\texttt{s(a).}}} \\
\text{\texttt{s(b).}} \\
\text{\texttt{t(c).}}
\]

Suppose we knew that all true instances of the relations \texttt{s(X)} and \texttt{t(Y)} were logical consequences of P1. That is to say, we know that all true instances of \texttt{s(X)} and \texttt{t(Y)} are logical consequences of P1.

\[
\text{\texttt{\texttt{t(c).}}} \\
\text{\texttt{\texttt{s(b).}}} \\
\text{\texttt{\texttt{s(a).}}}
\]

Therefore, we conclude that \texttt{s(c)} is false, or equivalently, \texttt{\texttt{not s(c).}} We are assured that \texttt{s(c)} is not a true instance of the relation \texttt{s(X)} and we know about the true instances of the relations \texttt{s(X)} and \texttt{t(Y)}.
Negation as Failure in Prolog

With the Closed World Assumption (CWA), we can implement negation as failure. In this case, we will allow the inference that \( \neg G \) is true if we fail to prove \( G \).

We cannot tell, in general, whether a goal \( G \) will succeed or fail in a finite amount of time. If \( G \) has a finitely failed search tree (one which is finite and has no success branches), we can define a negation as failure inference rule. This says that we may conclude \( \neg A \) if \( A \) has a finitely failed search tree.

A typical implementation:

\[
\text{not}(X) :- X, !, false.
\]

An example:

\[
\text{unmarried-student}(X) :- \text{not married}(X), \text{student}(X).
\]

Prolog's negation doesn't quite implement this. Why? Just because a goal \( G \) has a finitely failed search tree doesn't mean that the search tree which Prolog traverses is finitely failed (remember the literal order decodes the search tree structure). The order of the goals in the body of the rule is significant.

A more typical implementation:

\[
\text{not}(X) :- X, !, \text{false}.
\]

\[
\text{unmarried-student}(X) :- \text{not married}(X), \text{student}(X).
\]

We cannot tell, in general, whether a goal \( G \) will succeed or fail in a finite amount of time. If \( G \) has a finitely failed search tree, we can define a negation as failure inference rule. This says that we may conclude \( \neg A \) if \( A \) has a finitely failed search tree.

The inference that \( \neg G \) is true if we fail to prove \( G \) is valid within the Closed World Assumption (CWA). We can work correctly:

Negated goals must be ground for negation as failure in Prolog to work correctly.
Prolog is based on the Resolution method. To make it efficient, it accepts only definite clauses.

To make it efficient, it uses an unsound unification algorithm.

To make it efficient, it uses depth-first search.

To make it efficient, it doesn’t use factoring.

To make it efficient, it doesn’t use compilation.

To allow it to handle negative knowledge, the Closed World Assumption is taken and the Negation As Failure is implemented.

Many other techniques are also used: cut, compilation, ...