Knowledge Representation (KR)

- This material is covered in chapters 7—10 (R&N, 2nd ed) and chapters 7–9 and 12 (R&N, 3rd ed).
- Chapter 7 provides useful motivation for logic, and an introduction to some basic ideas. It also introduces propositional logic, which is a good background for first-order logic.
- What we cover here is mainly in Chapters 8 and 9. However, Chapter 8 contains some additional useful examples of how first-order knowledge bases can be constructed. Chapter 9 covers forward and backward chaining mechanisms for inference, while here we concentrate on resolution.
- Chapter 10 (2nd ed) (12 in 3rd ed) covers some of the additional notions that have to be dealt with when using knowledge representation in AI.

Some of the slides that follow are from Hector Levesque and Ron Brachman. We also acknowledge Craig Boutilier, Fahiem Bacchus, Sonya Alilin, Hojat Ghaderi, and others.

What is KNOWLEDGE?

Easier question: how do we talk about it?
- We say “John knows that ...” and fill in the blank (can be true/false, right/wrong)
- Contrast: “John fears that ...” (same content, different attitude)

Other forms of knowledge:
- know how, who, what, when, ...
- sensorimotor: typing, riding a bicycle
- affective: deep understanding

Belief: not necessarily true and/or held for appropriate reasons
Here we make no distinction between knowledge and belief

MAIN IDEA
Taking the world to be one way and not another

What is REPRESENTATION?

Symbols standing for things in the world

- first aid
- women
- John
- "John loves Mary" the proposition that John loves Mary

Knowledge Representation:
Symbolic encoding of propositions believed (by some agent)
What is REASONING?

Reasoning:
Manipulation of symbols encoding propositions to produce new representations of new propositions

Analogy: arithmetic "1011" + "10" → "1101"

\[
\begin{array}{c}
\text{eleven} \\
\text{two} \\
\text{thirteen}
\end{array}
\]

All men are mortals, socrates is a man. Thus, socrates is mortal
All foobars are bliffs, bebong is a foobar. Thus, bebong is a bliff

WHY Knowledge?

Taking an intentional stance (Daniel Dennet):
For sufficiently complex systems, it is sometimes compelling to describe systems in terms of mental properties -- beliefs, goals, fears, intentions

E.g., in a game-playing program
"because it believed its queen was in danger, but wanted to still control the centre of the board"

More useful than description about actual techniques used for deciding how to move
"because evaluation procedure P using minimax returned a value of +7 for this position."

Is KR just a convenient way of talking about complex systems?
- sometimes such anthropomorphizing is inappropriate (e.g., thermostat)
- can also be very misleading!
  fooling users into thinking a system knows more than it does

WHY Representation?

Taking Dennet’s intentional stance says nothing about what is or is not represented symbolically

  e.g., in game playing, perhaps the board position is represented but the goal of getting a knight out early is not

KR Hypothesis (Brian Smith)

"Any mechanically embodied intelligent process will be comprised of structural ingredients that a) we as external observers naturally take to represent a propositional account of the knowledge that the overall process exhibits, and b) independent of such external semantic attribution, play a formal but causal and essential role in engendering the behaviour that manifests that knowledge"

Two issues: existence of structures that
  * we can interpret propositionally
  * determine how the system behaves

KNOWLEDGE-BASED SYSTEM: one designed this way!

Two Examples

Both of the following systems can be described intentionally.

However, only the 2nd has a separate collection of symbolic structures (following the KR Hypothesis)

Example 2 has a knowledge base and thus the system is a knowledge-based system.

This is Prolog code, for those who don't know Prolog! (You don't need to know for the course.)

Example 1:

```
paintColour(snow) :- !, write("It's white.").
paintColour(grass) :- !, write("It's green.").
paintColour(sky) :- !, write("It's yellow.").
paintColour(X) :- !, write("Beats me.").
```

Example 2:

```
paintColour(X) :- colour(X,Y),!
  write("It's"). write(Y), write (".").
paintColour(X) :- !, write("Beats me.").
```

```
colour(snow,white).
colour(sky,yellow).
colour(X,Y) :- madeOf(X,Z), colour(Z,Y).
```

```
madeOf(grass,vegetation).
colour(vegetation, green).
```
**KR and AI**

Much of AI involves building systems that are knowledge based.

Ability derives in part from reasoning over explicitly represented knowledge:
- language understanding,
- planning,
- diagnosis, etc.

Some to a certain extent
- game-playing, vision, etc.

Some to a lesser extent
- speech, motor control, etc.

Current research question:
- how much of intelligent behaviour is knowledge-based?

---

**Why bother?**

Why not “compile out” knowledge into specialized procedures?
- distribute KB to procedures that need it (as in Example 1)
- almost always achieves better performance

No need to think. Just do it!
- riding a bike
- driving a car
- playing chess?
- doing math?
- staying alive?

Skills (Hubert Dreyfus)
- novices think, experts react
  - Compare to “expert systems” which are knowledge based.

---

**Advantage**

Knowledge-based system most suitable for open-ended tasks
- can structurally isolate reasons for particular behaviour

Good for:
- explanation and justification
  - “Because grass is a form of vegetation”
- informality: debugging the KB
  - “No the sky is not yellow. It’s blue.”
- extensibility: new relations (add new info)
  - “Canaries are yellow”
- extensibility: new applications (use your KB for different tasks)
  - Return a list of all the white things
  - Painting pictures

---

**Cognitive Penetrability**

Hallmark of knowledge-based system:
- The ability to be told facts about the world and adjust our behaviour correspondingly
  - E.g., read a book about canaries learn they’re yellow and add it to the KB

Cognitive penetrability (Zenon Pylyshyn)
- Actions that are conditioned by what is currently believed (by cognitive state)

An example:
- we normally leave the room if we hear a fire alarm
- we do not leave the room on hearing a fire alarm if we believe that the alarm is being tested/tempered
- we can come to this belief in very many ways
  - so this action is cognitively penetrable

A non-example:
- blinking reflex
Why reasoning?

Want knowledge to affect action
- do action $A$ if sentence $P$ is in KB
- do action $A$ if world believed in satisfies $P$

Difference:
- $P$ may not be explicitly represented
- Need to apply what is known in general to the particulars of the situation

Example:
- "patient $x$ is allergic to medication $m$."
- "Anybody allergic to medication $m$ is also allergic to $n$."
- Is it OK to prescribe $n$ for $x$?
- Usually need more than just DB-style retrieval of facts in the KB!

Using Logic

No universal language/semantics
- Why not English?
- Different tasks/worlds
- Different ways to carve up the world

No universal reasoning scheme
- Geared to language
- Sometimes want "extralogical" reasoning

Start with first-order predicate calculus (FOL)
- Invented by philosopher Frege for the formalization of mathematics

Why Knowledge Representation? An Example

- Consider the task of understanding a simple story.
- How do we test understanding?
- Not easy, but understanding at least entails some ability to answer simple questions about the story.
Example.

- Three little pigs: Mother sends them to "seek their fortune"

1st pig builds a house of straw
2nd pig builds a house of sticks
3rd pig builds a house of bricks

Example.

- Three little pigs

Wolf blows down the straw house, and eats the pig!
Wolf blows down the sticks house, and eats the pig!
Wolf cannot huff and puff the brick house!

Example

- Why couldn't the wolf blow down the house made of bricks?
- What background knowledge are we applying to come to that conclusion?
  - Brick structures are stronger than straw and stick structures.
  - Objects, like the wolf, have physical limitations. The wolf can only blow so hard.

Why Knowledge Representation?

- Large amounts of knowledge are used to understand the world around us, and to communicate with others.
- We also have to be able to reason with that knowledge
  - Our knowledge won't be about the blowing ability of wolves in particular, it is about physical limits of objects in general.
  - We have to employ reasoning to make conclusions about the wolf.
  - More generally, reasoning provides an exponential or more compression in the knowledge we need to store. I.e., without reasoning we would have to store an infeasible amount of information: e.g., Elephants can't fit into teacups.
Logical Representations

• AI typically employs logical representations of knowledge.

• Logical representations useful for a number of reasons:
  - They are mathematically precise, thus we can analyze their limitations, their properties, the complexity of inference etc.
  - They are formal languages, thus computer programs can manipulate sentences in the language.
  - They come with both a formal syntax and a formal semantics.
  - Typically, have well developed proof theories: formal procedures for reasoning (achieved by manipulating sentences).

The Knowledge Base

• The Knowledge Base is a set of sentences.
  - Syntactically well-formed
  - Semantically meaningful

• A user can perform two actions to the KB:
  - Tell the KB a new fact
  - Ask the KB a question

Syntax of Sentences

English acceptable an one is sentence This.

vs.

This English sentence is an acceptable one.

\[ \lor P \neg \land Q \land R \]

vs.

\[ P \lor \neg Q \land R \]
Semantics of Sentences

This hungry classroom is a jobless moon.

- Why is this syntactically correct sentence not meaningful?

\[ P \lor (\neg Q \land R) \]

- Represents a world where either P is true, or Q is not true and R is true.

Entailments

\[ \alpha \models \beta \]

- read as “\( \alpha \) entails \( \beta \)”, or “\( \beta \) follows logically from \( \alpha \)”
- meaning that in any world in which \( \alpha \) is true, \( \beta \) is true as well.

For example

\[ (P \land Q) \models (P \lor R) \]

Syntactical Derivation

\[ \alpha \vdash \beta \]

- read as “\( \alpha \) derives \( \beta \)”
- meaning “from sentence \( \alpha \), following the syntactical derivation rules, we can obtain sentence \( \beta \)”

For example

\[ \neg (A \lor B) \vdash (\neg A \land \neg B) \]

KNOWLEDGE REPRESENTATION AND REASONING
The Reasoning Process

Desired Properties of Reasoning

Soundness
- If $\text{KB} \vdash f$ then $\text{KB} \models f$
- i.e. all conclusions arrived at via the proof procedure are correct: they are logical consequences.

Completeness
- If $\text{KB} \models f$ then $\text{KB} \vdash f$
- i.e. every logical consequence can be generated by the proof procedure.

First-Order Logic (FOL)

Propositional logic assumes the world contains facts.

First-order logic (like natural language) assumes the world contains

- **Objects**: people, houses, numbers, colors, baseball games, wars, …
- **Relations**: red, round, prime, brother of, bigger than, part of, comes between, …
- **Functions**: father of, best friend, one more than, plus, …

First-Order Logic

**Syntax**: A grammar specifying what are legal syntactic constructs of the representation.

**Semantics**: A formal mapping from syntactic constructs to set theoretic assertions.
Syntax of FOL: Basic elements

- Constants: sheila, 2, blockA,...
- Functions: sqrt, brother,...
- Variables: x, y, a, b,...
- Predicates: colour, on, brotherOf, >,...
- Connectives: ¬, →, ∧, ∨, ≡
- Equality: =
- Quantifiers: ∀, ∃

Propositions are statements about the world and are either true/false.

Functions evaluate to a value. E.g., \( \sqrt{4} = 2 \) and \( \text{Fncity(Dave)} = \text{Ottawa} \).

Syntax of Propositional Logic: Basic elements

- Constants: sheila, 2, blockA,...
- Functions: sqrt, leftLegOf,...
- Variables: x, y, a, b,...
- Connectives: ¬, →, ∧, ∨, ≡
- Propositions: a-ison-b, b-ison-c, ...
- Equality: =
- Quantifiers: ∀, ∃

The terms are either:
- a variable
- a constant
- an expression of the form \( f(t_1, \ldots, t_k) \) where
  (a) \( f \) is a function symbol;
  (b) \( k \) is its arity;
  (c) each \( t_i \) is a term

Note:
- Constants are functions taking zero arguments.
- Use UPPER CASE for variables, lower case for function/constant/predicate symbols.

First Order Syntax

Start with a set of primitive symbols:
- constant symbols.
- function symbols.
- predicate symbols (for predicates and relations).
- variables.

Each function and predicate symbol has a specific arity (determines the number of arguments it takes).

First Order Syntax—Building up.

A term is either:
- a variable
- a constant
- an expression of the form \( f(t_1, \ldots, t_k) \) where
  (a) \( f \) is a function symbol;
  (b) \( k \) is its arity;
  (c) each \( t_i \) is a term

Note:
- constants are functions taking zero arguments.
- Use UPPER CASE for variables, lower case for function/constant/predicate symbols.
First Order Syntax—Building up.

An atom is

An expression of the form $p(t_1, \ldots, t_k)$ where

- $p$ is a predicate symbol;
- $k$ is its arity;
- each $t_i$ is a term

Semantic Intuition (formalized later)

Terms denote individuals:

- constants denote specific individuals;
- functions map tuples of individuals to other individuals
  - $\text{bill, jane, father(jane), father(father(jane))}$
  - $X, \text{father(X), hotel7, rating(hotel7), cost(hotel7)}$

Atoms denote facts that can be true or false about the world:

- $\text{father_of(jane, bill), female(jane), system_down()}$
- $\text{satisfied(client15), satisfied(C)}$
- $\text{desires(client15, rome, week29), desires(X, Y, Z)}$
- $\text{rating(hotel7, 4), cost(hotel7, 125)}$

First Order Syntax—Building up.

Atoms are formulas. (Atomic formulas).

- The negation (NOT) of a formula is a new formula $\neg f$
  
  Asserts that $f$ is false.

- The conjunction (AND) of a set of formulas is a formula.
  $f_1 \land f_2 \land \ldots \land f_n$ where each $f_i$ is formula
  
  Asserts that each formula $f_i$ is true.

First Order Syntax—Building up.

- The disjunction (OR) of a set of formulas is a formula.
  $f_1 \lor f_2 \lor \ldots \lor f_n$ where each $f_i$ is formula
  
  Asserts that at least one formula $f_i$ is true.

- Existential Quantification $\exists$.
  
  $\exists X. f$ where $X$ is a variable and $f$ is a formula.
  
  Asserts there is some individual such that $f$ under than binding will be true.

- Universal Quantification $\forall$.
  
  $\forall X. f$ where $X$ is a variable and $f$ is a formula.
  
  Assets that $f$ is true for every individual.
First Order Syntax—abbreviations.

- Implication (→):
  \[ f_1 \rightarrow f_2 \text{ is equivalent to } \neg f_1 \lor f_2. \]

Semantics.

- Formulas (syntax) can be built up recursively, and can become arbitrarily complex.
- Intuitively, there are various distinct formulas (viewed as strings) that really are asserting the same thing
  - \( \forall X, Y. \text{ elephant}(X) \land \text{teacup}(Y) \rightarrow \text{largerThan}(X,Y) \)
  - \( \forall X, Y. \text{teacup}(Y) \land \text{elephant}(X) \rightarrow \text{largerThan}(X,Y) \)
- To capture this equivalence and to make sense of complex formulas we utilize the semantics.

Semantics.

- A formal mapping from formulas to semantic entities (individuals, sets and relations over individuals, functions over individuals).
- The mapping mirrors the recursive structure of the syntax, so we can give any formula, no matter how complex a mapping to semantic entities.

Semantics—Formal Details

First, we must fix the particular first-order language we are going to provide semantics for. The primitive symbols included in the syntax defines the particular language. \( L(F, P, V) \)

- \( F = \text{set of function (and constant symbols)} \)
  - \( \text{Each symbol } f \text{ in } F \text{ has a particular arity.} \)
- \( P = \text{set of predicate symbols} \)
  - \( \text{Each symbol } p \text{ in } P \text{ has a particular arity.} \)
- \( V = \text{an infinite set of variables} \).
Semantics—Formal Details

An interpretation (model) is a tuple
\[(D, \Phi, \Psi, v)\]

- \(D\) the domain of discourse, is a non-empty set (domain of individuals)
- \(\Phi\) is a mapping: \(\Phi(f) \rightarrow (D^k \rightarrow D)\)
  - maps k-ary function symbol \(f\), to a function from k-ary tuples of individuals to individuals.
- \(\Psi\) is a mapping: \(\Psi(p) \rightarrow (D^k \rightarrow \text{True/False})\)
  - maps k-ary predicate symbol \(p\), to an indicator function over k-ary tuples of individuals (a subset of \(D^k\))
- \(v\) is a variable assignment function. \(v(X) = d \in D\) (it maps every variable to some individual)

Intuitions: Domain of Discourse

- Domain \(D\): \(d \in D\) is an individual
  
  E.g., \{malaria, flu, jane, ache, fever, antibiotic, quarantine, 100, 60, 37…\}

- Underlined symbols denote domain individuals (as opposed to symbols of the first-order language)
- Domains often infinite, but we’ll use finite models to prime our intuitions

Intuitions: \(\Phi\) (individual denoted by fn.)

\(\Phi(f) \rightarrow (D^k \rightarrow D)\)
Given k-ary function \(f\), \(k\) individuals,
what individual does \(f(d_1, ..., d_k)\) denote

- 0-ary functions (constants) are mapped to specific individuals in \(D\).
  - \(\Phi(\text{client17}) = \text{craig}, \Phi(\text{hotel5}) = \text{le-fleabag}, \Phi(\text{rome}) = \text{rome}\)
- 1-ary functions are mapped to functions in \(D \rightarrow D\)
  - \(\Phi(\text{minquality}) = f_{\text{minquality}}:\ f_{\text{minquality}}(\text{craig}) = 3\text{stars}\)
  - \(\Phi(\text{rating}) = f_{\text{rating}}:\ f_{\text{rating}}(\text{grandhotel}) = 5\text{stars}\)
- 2-ary functions are mapped to functions from \(D^2 \rightarrow D\)
  - \(\Phi(\text{distance}) = f_{\text{distance}}:\ f_{\text{distance}}(\text{toronto}, \text{sienna}) = 3256\)
- \(n\)-ary functions are mapped similarly.
Intuitions: $\Psi$ (truth or falsity of formula)

$\Psi(p) \rightarrow (D \rightarrow \text{True}/\text{False})$

Given $k$-ary predicate, $k$ individuals,

does the relation denoted by $p$ hold of these?

$\Psi(p)(<d_1, \ldots, d_k>) = \text{true}$?

• 0-ary predicates are mapped to true or false.

  $\Psi(\text{rainy}) = \text{True}$  
  $\Psi(\text{sunny}) = \text{False}$

• 1-ary predicates are mapped indicator functions of subsets of $D$.

  $\Psi(\text{satisfied}) = p_{\text{satisfied}}$: 
  
  $p_{\text{satisfied}}(\text{craig}) = \text{True}$

  $p_{\text{satisfied}}(\text{le-fleabag}) = \text{False}$

• 2-ary predicates are mapped to indicator functions over $D^2$

  $\Psi(\text{location}) = p_{\text{location}}$: 
  
  $p_{\text{location}}(\text{grandhotel, rome}) = \text{True}$

  $p_{\text{location}}(\text{grandhotel, sienna}) = \text{False}$

• $n$-ary predicates

Intuitions: $v$

$v$ exists to take care of quantification.

As we will see the exact mapping it specifies will not matter.

Notation: $v[X/d]$ is a new variable assignment function.

• Exactly like $v$, except that it maps the variable $X$ to the
  individual $d$.

• Maps every other variable exactly like $v$:

  $v'(Y) = v[X/d](Y)$

Semantics—Building up

Given language $L(F, P, V)$, and an interpretation $I = \langle D, \Psi, v \rangle$

a) Constant $c$ (0-ary function) denotes an individual

$I(c) = \Phi(c) \in D$

b) Variable $X$ denotes an individual

$I(X) = v(X) \in D$ (variable assignment function).

c) Term $t = f(t_1, \ldots, t_k)$ denotes an individual

$I(t) = \Phi(f)(l(t_1), \ldots, l(t_k)) \in D$

We recursively find the denotation of each term, then we
apply the function denoted by $f$ to get a new individual.

Hence terms always denote individuals under
an interpretation $I$
Semantics—Building up

Formulas

a) atom \( a = p(t_1, \ldots, t_k) \) has truth value
\[ I(a) = \Psi(p)(I(t_1), \ldots, I(t_k)) \in \{ \text{True, False} \} \]

We recursively find the individuals denoted by the \( t_i \), then we check to see if this tuple of individuals is in the relation denoted by \( p \).

Semantics—Building up

Formulas

b) Negated formulas \( \neg f \) has truth value
\[ I(\neg f) = \text{True if } I(f) = \text{False} \]
\[ I(\neg f) = \text{False if } I(f) = \text{True} \]

c) And formulas \( f_1 \land f_2 \land \ldots \land f_n \) have truth value
\[ I(f_1 \land f_2 \land \ldots \land f_n) = \text{True if every } I(f_i) = \text{True}. \]
\[ I(f_1 \land f_2 \land \ldots \land f_n) = \text{False otherwise.} \]

d) Or formulas \( f_1 \lor f_2 \lor \ldots \lor f_n \) have truth value
\[ I(f_1 \lor f_2 \lor \ldots \lor f_n) = \text{True if any } I(f_i) = \text{True}. \]
\[ I(f_1 \lor f_2 \lor \ldots \lor f_n) = \text{False otherwise.} \]

e) Existential formulas \( \exists X. f \) have truth value
\[ I(\exists X. f) = \text{True if there exists a } d \in D \text{ such that } \]
\[ I'(f) = \text{True} \]
where \( I' = \langle D, \Phi, \Psi, v[X/d] \rangle \)
\[ \text{False otherwise.} \]

I’ is just like I except that its variable assignment function now maps X to d. “d” is the individual of which “f” is true.

g) Universal formulas \( \forall X. f \) have truth value
\[ I(\forall X. f) = \text{True if for all } d \in D \]
\[ I'(f) = \text{True} \]
where \( I' = \langle D, \Phi, \Psi, v[X/d] \rangle \)
\[ \text{False otherwise.} \]

Now “f” must be true of every individual “d”.

Hence formulas are always either True or False under an interpretation I
Example

\[ D = \{ \text{bob}, \text{jack}, \text{fred} \} \]
\[ I(\forall X. \text{happy}(X)) \]

1. \( \Psi(\text{happy})(\forall[X/\text{bob}](X)) = \Psi(\text{happy})(\text{bob}) = \text{True} \)
2. \( \Psi(\text{happy})(\forall[X/\text{jack}](X)) = \Psi(\text{happy})(\text{jack}) = \text{True} \)
3. \( \Psi(\text{happy})(\forall[X/\text{fred}](X)) = \Psi(\text{happy})(\text{fred}) = \text{True} \)

Therefore \( I(\forall X. \text{happy}(X)) = \text{True} \).

Models—Examples.

### Language (Syntax)
- Constants: a, b, c, e
- Functions:
  - No function
- Predicates:
  - on: binary
  - above: binary
  - clear: unary
  - ontoable: unary

### Environment
- Constants: a, b, c, e
- Predicates:
  - on (binary)
  - above (binary)
  - clear (unary)
  - ontoable (unary)

Think of it as possible way the world could be.
Aside on Notation.

Model I

| D = \{A, B, C, E\} |
| Φ(a) = A, Φ(b) = B, Φ(c) = C, Φ(e) = E |
| Ψ(on) = \{(A,B),(B,C)\} |
| Ψ(above) = \{(A,B),(B,C),(A,C)\} |
| Ψ(clear) = \{A,E\} |
| Ψ(ontable) = \{C,E\} |

Comment on notation:
To this point we have represented an interpretation as a tuple, \((D, Φ, Ψ, v)\). It is also common practice to abbreviate this and refer to an interpretation function as \(s\) and to denote the application of the interpretation function by a superscripting of \(s\).

E.g.,
\(a^s = A\), \(b^s = B\) etc.
\(on^s = \{(A,B),(B,C)\}\)
\(above^s = \{(A,B),(B,C),(A,C)\}\)
\(clear^s = \{A,E\}\)
\(ontable^s = \{C,E\}\)

Models—Formulas true or false?

Model I

∀X,Y. on(X,Y)→above(X,Y)

∀X,Y. above(X,Y)→on(X,Y)

Models—Examples.

Model I

∀X∃Y. (clear(X) ∨ on(Y,X))

∀X∃Y. (clear(X) ∨ on(Y,X))

KB—many models

KB

1. on(b,c)
2. clear(e)
KB—many models

KB
1. on(b,c)
2. clear(e)

Models

• Let our knowledge base KB consist of a set of formulas.

• We say that I is a model of KB or that I satisfies KB
  • If, every formula \( f \in KB \) is true under I

• We write \( I \models KB \) if I satisfies KB, and \( I \models f \) if \( f \) is true under I.

What’s Special About Models?

• When we write KB, we have an intended interpretation for it — a way that we think “the world” — the scenario we’re modeling -- will be.

• This means that every statement in KB is true in “the world”.

• Note however, that not every thing true in the world need be contained in KB. We might have only incomplete knowledge.

Models support reasoning.

Suppose formula \( f \) is not mentioned in KB, but is true in every model of KB, i.e.,

\[
\text{For all } I, \text{ if } I \models KB \text{ then } I \models f.
\]

Then we say that \( f \) is a logical consequence of KB or that KB entails \( f \)

\[
KB \models f.
\]

Since “the world” is a model of KB, \( f \) must be true in “the world”.

This means that entailment is a way of finding new true facts that were not explicitly mentioned in KB.

QUESTION: If KB doesn’t entail \( f \), is \( f \) false in “the world”?
Models Graphically (propositional example)

Propositional KB: \( a, c \rightarrow b, b \rightarrow c, d \rightarrow b, \neg b \rightarrow \neg c \)

Set of All Interpretations

\[
\begin{align*}
\text{Models of KB} \\
\{ & a, b, c, d \\
\{ & a, \neg b, \neg c, \neg d \\
\{ & a, b, c, \neg d
\end{align*}
\]

Axiomatizing the Domain

The more sentences in KB, the fewer models (satisfying interpretations) there are.

The more you write down (as long as it’s all true!), the “closer” you get to a complete description of “the world” and the less is left to doubt, because each sentence in KB rules out certain unintended interpretations.

Writing down the description of the world is called \textit{axiomatizing the domain.}

We will construct a knowledge base by writing down a description of (some of) the world in first-order logic.

Computing logical consequences

We want procedures for computing logical consequences that can be implemented in our programs.

This would allow us to reason about the world. In particular we would:
1. Represent aspects of our knowledge of the world as logical formulas
2. Apply procedures for generating logical consequences

Procedures for computing logical consequences are called \textit{proof procedures.}

Proof Procedures

- Proof procedures work by simply manipulating formulas. They do not know or care anything about interpretations.

- Nevertheless they respect the semantics of interpretations!

- We will employ a proof procedure for first-order logic called resolution.
  - Resolution is the mechanism used by PROLOG
Properties of Proof Procedures

Before presenting the details of resolution, we want to look at properties we would like to have in a (any) proof procedure.

We write $KB \vdash f$ to indicate that $f$ can be proved from $KB$.

The proof procedure used is implicit. Many different proof procedures exist. Again, we will be studying resolution.

Properties of Proof Procedures

**Soundness**

If $KB \vdash f$ then $KB \models f$

i.e., all conclusions arrived at via the proof procedure are correct; they are logical consequences.

**Completeness**

If $KB \models f$ then $KB \vdash f$

i.e., every logical consequence can be generated by the proof procedure.

Note proof procedures are computable, but they might have very high complexity in the worst case. So completeness is not necessarily achievable in practice.

Resolution

Resolution is a rule of inference leading to a theorem proving technique in propositional and first-order logic.

Iteratively applying the resolution rule allows us to determine whether a propositional formula is satisfiable or a first-order formula is unsatisfiable.

**Resolution Rule**

From the two clauses

$P \lor Q$

$
eg P$

We infer the new clause

$Q$

The original algorithm required ground instances of formulas. What's the problem with this?

The unification algorithm allows "instantiation or grounding on demand" to preserve refutation completeness.

Clausal form

Resolution works with formulas expressed in clausal form.

- A literal is an atomic formula or the negation of an atomic formula.
- dog(fido), ¬cat(fido)
- A clause is a disjunction of literals:
  - ¬owns(fido,fred) ∨ ¬dog(fido) ∨ person(fred)
  - We write
    - (¬owns(fido,fred), ¬dog(fido), person(fred))
- A ground clause is a clause containing no variables e.g., the clause dog(X) ∨ person(fred) is not ground.
  - The clause dog(fido) ∨ person(fred) is ground.
- A clausal theory is a conjunction of clauses.
Resolution Rule for Ground Clauses

The resolution proof procedure consists of only
one simple rule:

From the two clauses
• (P, Q1, Q2, ..., Qk)
• (¬P, R1, R2, ..., Rn)
We infer the new clause
• (Q1, Q2, ..., Qk, R1, R2, ..., Rn)

Example:
• (¬largerThan(clyde,cup), ¬fitsIn(clyde,cup))
• (fitsIn(clyde,cup))
Infer ¬largerThan(clyde,cup)

Resolution Proof: Forward chaining

Logical consequences can be generated from the
resolution rule in two ways:

1. Forward Chaining Inference ("Consequence Finding")*
   • If we have a sequence of clauses C1, C2, ..., Ck
   • Such that each Ci is either in KB or is the result of a
     resolution step involving two prior clauses in the
     sequence.
   • We then have that KB ⊨ CK.
   Forward chaining is sound so we also have KB ⊢ CK
   * The meat grinder

Resolution Proof: Refutation proofs

2. Refutation Proofs.
   • We determine if KB ⊨ f by showing that a contradiction
     can be generated from KB ∧ ¬f.
   • In this case a contradiction is an empty clause ( ).
   • We employ resolution to construct a sequence of
     clauses C1, C2, ..., Cm such that
     • Ci is in KB ∧ ¬f, or is the result of resolving two
       previous clauses in the sequence.
     • Cm = () i.e. its the empty clause.

Resolution Proof: Refutation proofs

If we can find a sequence C1, C2, ..., Cm=(), we have that
• KB ⊨ f
Furthermore, this procedure is sound so
• KB ⊨ f

The procedure is also complete (refutation complete) so it is
 capable of finding a proof of any f that is a logical
 consequence of KB. i.e.

• If KB ⊨ f then we can generate a refutation from KB ∧ ¬f
Resolution Proofs Example

Want to prove likes(clyde,peanuts) from:
- (elephant(clyde), giraffe(clyde)) [1]
- (~elephant(clyde), likes(clyde,peanuts)) [2]
- (~giraffe(clyde), likes(clyde,leaves)) [3]
- ~likes(clyde,leaves) [4]

Approach 1: Forward Chaining Proof:
- [3&4] ~giraffe(clyde) [5]
- [5&1] elephant(clyde) [6]
- [6&2] likes(clyde,peanuts) [7] ✓

Resolution Proofs

Proofs by refutation are generally easier to find. They're more focused to the particular conclusion we are trying to reach.

To develop a resolution proof procedure for First-Order logic we need:
1. A way of converting KB and f (the query) into clausal form.
2. A way of doing resolution even when we have variables. This is called unification.

Conversion to Clausal Form

To convert the KB into Clausal form we perform the following 8-step procedure:

1. Eliminate Implications.
3. Standardize Variables.
4. Skolemize.
5. Convert to Prenex Form.
6. Distribute disjunctions over conjunctions.
7. Flatten nested conjunctions and disjunctions.
8. Convert to Clauses.
C-T-C-F: Eliminate implications

We use this example to show each step:
\[ \forall x. p(x) \to ( \forall y. p(y) \to p(f(x,y)) \land \neg(\forall y. \neg q(x,y) \land p(y)) ) \]

1. Eliminate implications: \( A \to B \to \neg A \lor B \)
\[ \forall x. \neg p(x) \lor ( \forall y. \neg p(y) \lor p(f(x,y)) \land \neg(\forall y. \neg q(x,y) \land p(y)) ) \]

C-T-C-F: Move \( \neg \) Inwards

\[ \forall x. \neg p(x) \lor ( \forall y. \neg p(y) \lor p(f(x,y)) \land \neg(\forall y. \neg q(x,y) \land p(y)) ) \]

2. Move Negations Inwards (and simplify \( \neg \neg \))
\[ \forall x. \neg p(x) \lor ( \forall y. \neg p(y) \lor p(f(x,y)) \land \exists y. q(x,y) \lor \neg p(y) ) \]

C-T-C-F: Standardize Variables

\[ \forall x. \neg p(x) \lor ( \forall y. \neg p(y) \lor p(f(x,y)) \land \exists y. q(x,y) \lor \neg p(y) ) \]

3. Standardize Variables (Rename variables so that each quantified variable is unique)
\[ \forall x. \neg p(x) \lor ( \forall y. \neg p(y) \lor p(f(x,y)) \land \exists z. q(x,z) \lor \neg p(z) ) \]
C-T-C-F: Skolemize

\[ \forall X. \neg p(X) \lor (\forall Y. \neg p(Y) \lor p(f(X,Y)) \land \exists Z. q(X,Z) \lor \neg p(Z)) \]

4. Skolemize (Remove existential quantifiers by introducing new function symbols).

\[ \forall X. \neg p(X) \lor (\forall Y. \neg p(Y) \lor p(f(X,Y)) \land q(X,g(X)) \lor \neg p(g(X))) \]

C-T-C-F: Skolemization continued…

Consider \( \exists Y. \text{elephant}(Y) \land \text{friendly}(Y) \)

• This asserts that there is some individual (binding for \( Y \)) that is both an elephant and friendly.

• To remove the existential, we invent a name for this individual, say \( a \). This is a new constant symbol not equal to any previous constant symbols to obtain:

\[ \text{elephant}(a) \land \text{friendly}(a) \]

• This is saying the same thing, since we do not know anything about the new constant \( a \). Recall that a constant is a 0-ary function symbol.

C-T-C-F: Skolemization continue

• It is essential that the introduced symbol “a” is new. Else we might know something else about “a” in KB.

• If we did know something else about “a” we would be asserting more than the existential.

• In the original quantified formula we know nothing about the variable “Y”. Just what was being asserted by the existential formula.

Now consider \( \forall X. \exists Y. \text{loves}(X,Y) \).

• This formula claims that for every \( X \) there is some \( Y \) that \( X \) loves (perhaps a different \( Y \) for each \( X \)).

• Replacing the existential by a new constant won’t work \( \forall X. \text{loves}(X,a) \).

Because this asserts that there is a particular individual “a” loved by every \( X \).

• To properly convert existential quantifiers scoped by universal quantifiers we must use functions not just constants.
Recall we are considering the example

\[ \forall X. Y. \text{loves}(X,Y) \]

- We must use a function that mentions every universally quantified variable that scopes the existential.
- In this case X scopes Y so we must replace the existential Y by a function of X

\[ \forall X. \text{loves}(X,g(X)) \]

where g is a new function symbol.

This formula asserts that for every X there is some individual (given by g(X)) that X loves. g(X) can be different for each different binding of X.

---

**C-T-C-F: Skolemization Examples**

- \[ \forall X. \exists Y. \forall Z. \exists W. \text{loves}(X,Y,Z,W) \Rightarrow ? \]

- \[ \forall X. \exists Y. \forall Z. \exists W. \text{loves}(X,Y,g(W)) \Rightarrow ? \]

- \[ \forall X. \exists Y. \forall Z. \exists W. \text{loves}(X,Y,Z,W) \land q(Z,W) \Rightarrow ? \]

---

**C-T-C-F: Convert to prenex**

- \[ \forall X. \neg p(X) \lor \left( \forall Y. \neg p(Y) \lor p(f(X,Y)) \land q(X,g(X)) \lor \neg p(g(X)) \right) \]

5. Convert to prenex form. (Bring all quantifiers to the front—only universals, each with different name).

- \[ \forall X. \forall Y. \neg p(X) \lor \left( \neg p(Y) \lor p(f(X,Y)) \land q(X,g(X)) \lor \neg p(g(X)) \right) \]
**C-T-C-F: disjunctions over conjunctions**

\[ \forall X \forall Y. \neg p(X) \lor (\neg p(Y) \lor p(f(X,Y)) \land q(X,g(X)) \lor \neg p(g(X))) \]

6. Disjunction over Conjunction

\[ A \lor (B \land C) \Rightarrow (A \lor B) \land (A \lor C) \]

\[ \forall XY. \neg p(X) \lor \neg p(Y) \lor p(f(X,Y)) \land \neg p(X) \lor q(X,g(X)) \lor \neg p(g(X)) \]

---

**Unification**

- Ground clauses are clauses with no variables in them. For ground clauses we can use syntactic identity to detect when we have a \( P \) and \( \neg P \) pair.

- What about variables? Can the clauses
  
  \( (P(\text{john}), Q(\text{fred}), R(X)) \)
  
  \( (\neg P(Y), R(\text{susan}), R(Y)) \)

  be resolved?

---

**C-T-C-F: flatten & convert to clauses**

7. Flatten nested conjunctions and disjunctions.

\[ (A \lor (B \lor C)) \Rightarrow (A \lor B \lor C) \]

8. Convert to Clauses

(removes quantifiers and break apart conjunctions).

\[ \forall XY. \neg p(X) \lor \neg p(Y) \lor p(f(X,Y)) \land \neg p(X) \lor q(X,g(X)) \lor \neg p(g(X)) \]

\[ a) \quad \neg p(X) \lor \neg p(Y) \lor p(f(X,Y)) \]

\[ b) \quad \neg p(X) \lor q(X,g(X)) \lor \neg p(g(X)) \]

---

**Unification.**

- Intuitively, once reduced to clausal form, all remaining variables are universally quantified. So, implicitly the clause

\( (\neg P(Y), R(\text{susan}), R(Y)) \)

represents clauses like

\( (\neg P(\text{fred}), R(\text{susan}), R(\text{fred})) \)

\( (\neg P(\text{john}), R(\text{susan}), R(\text{john})) \)

- So there is a "specialization" of \( (\neg P(Y), R(\text{susan}), R(Y)) \) that can be resolved w/ (a specialization of) \( (P(\text{john}), Q(\text{fred}), R(X)) \) from the prev page

- In particular,

\( (\neg P(\text{john}), R(\text{susan}), R(\text{john})) \) can be resolved with

\( (P(\text{john}), Q(\text{fred}), R(\text{john})) \) producing the new clause

\( (R(\text{susan}), R(\text{john}), Q(\text{fred})) \)
Unification.

• We want to be able to match conflicting literals, even when they have variables. This matching process automatically determines whether or not there is a "specialization" that matches.

• But, we don't want to over specialize!

Consider the following example

\(\neg p(X), s(X), q(fred)\)

\((p(Y), r(Y))\)

We need to make \(\neg p(X)\) and \(p(Y)\) look the same so we can resolve them away, producing a new clause \((s(?), q(fred), r(??))\).

How do we do this?

Possible resolvants

• \((s(john), q(fred), r(john))\) \(\{Y=\text{john}, X=\text{john}\}\)
• \((s(sally), q(fred), r(sally))\) \(\{Y=\text{sally}, X=\text{sally}\}\)
• \((s(X), q(fred), r(X))\) \(\{Y=X\}\)

The last resolvant is "most-general", the other two are specializations.

We want the most general clause for use in future resolution steps.

Unification

Unification is a mechanism for finding a "most general" matching.

A key component of unification is substitution.

• A substitution is a finite set of equations of the form

\((V = t)\)

where \(V\) is a variable and \(t\) is a term not containing \(V\). (\(t\) might contain other variables).

Substitutions.

• We can apply a substitution \(\sigma\) to a formula \(\alpha\) to obtain a new formula \(\alpha\sigma\) by simultaneously replacing every variable mentioned in the left hand side of the substitution by the right hand side.

E.g., substituting variable \(Y\) for \(X\) and \(f(a)\) for variable \(Y\):

\(p(X, g(Y,Z))(X=Y, Y=f(a)) \Rightarrow p(Y, g(f(a),Z))\)

• IMPORTANT: Note that the substitutions are not applied sequentially, i.e., the first \(Y\) is not subsequently replaced by \(f(a)\).
**Substitutions:** Composition of Substitutions

We can compose two substitutions \( \theta \) and \( \sigma \) to obtain a new substitution \( \theta \sigma \). (This is sequential.)

Let \( \theta = \{ X_1 = s_1, X_2 = s_2, \ldots, X_m = s_m \} \)
\[ \sigma = \{ Y_1 = t_1, Y_2 = t_2, \ldots, Y_k = t_k \} \]

Step 1 to compute the composition \( \theta \sigma \)
- we apply \( \sigma \) to each right-hand side of \( \theta \), and then
- add all of the equations of \( \sigma \).

Resulting in:

\[ \mathcal{S} = \{ X_1 = s_1 s, X_2 = s_2 s, \ldots, X_m = s_m s, Y_1 = t_1, Y_2 = t_2, \ldots, Y_k = t_k \} \]

---

**Composition Example**

\( \theta = \{ X = f(Y), Y = Z \} \), \( \sigma = \{ X = a, Y = b, Z = Y \} \)

Compute \( \theta \sigma \)

1. Construct the composition as on the last page
   \[ \mathcal{S} = \{ X = f(Y), Y = Z, X = a, Y = b, Z = Y \} \]
2. Delete any identities, i.e., equations of the form \( V = V \).
3. Delete any equation \( Y = s \), where \( Y \) is equal to one of the \( X \) in \( \theta \).

Step 1
\[ \mathcal{S} = \{ X = f(Y), X = a, Y = b, Z = Y \} \]

Step 2
\[ \mathcal{S} = \{ X = f(b), X = a, Y = b, Z = Y \} \]

Step 3
\[ \mathcal{S} = \{ X = f(b), Y = b, Z = Y \} \]

---

**Substitutions:**

- The empty substitution \( \varepsilon = \{ \} \) is also a substitution, and it acts as an identity under composition.

- More importantly substitutions when applied to formulas are associative:
  \[ (f \theta) \sigma = f(\theta \sigma) \]

- **Composition** is simply a way of converting the sequential application of a series of substitutions to a single simultaneous substitution.
Unifiers

• A unifier of two formulas $f$ and $g$ is a substitution $\sigma$ that makes $f$ and $g$ syntactically identical.

• Not all formulas can be unified—substitutions only affect variables.

\[
p(f(X), a) \quad p(Y, f(w))
\]

• This pair cannot be unified as there is no way of making $a = f(w)$ with a substitution.

MGU

A substitution $\sigma$ of two formulas $f$ and $g$ is a Most General Unifier (MGU) if

1. $\sigma$ is a unifier.
2. For every other unifier $\theta$ of $f$ and $g$ there must exist a third substitution $\lambda$ such that

\[
\theta = \sigma\lambda.
\]

This says that every other unifier is "more specialized" than $\sigma$. The MGU of a pair of formulas $f$ and $g$ is unique up to renaming.

MGU

\[
p(f(X), Z) \quad p(Y, a)
\]

1. $\sigma = \{Y = f(a), X = a, Z = a\}$ is a unifier.

\[
p(f(X), Z)^\sigma = p(Y, a)^\sigma =
\]

But it is not an MGU.

2. $\theta = \{Y = f(X), Z = a\}$ is an MGU.

\[
p(f(X), Z)^\theta = p(Y, a)^\theta =
\]

But it is not an MGU.

MGU

\[
p(f(X), Z) \quad p(Y, a)
\]

1. $\sigma = \{Y = f(a), X = a, Z = a\}$ is a unifier.

\[
p(f(X), Z)^\sigma = p(f(a), a)
\]

But it is not an MGU.

2. $\theta = \{Y = f(X), Z = a\}$ is an MGU.

\[
p(f(X), Z)^\theta = p(Y, a)^\theta =
\]
1. \( \sigma = \{ Y = f(a), X=a, Z=a \} \) is a unifier.
   
   \[
   \begin{align*}
   p(f(X), Z) &\sigma = p(f(a), a) \\
   p(Y, a) &\sigma = p(f(a), a)
   \end{align*}
   \]
   
   But it is not an MGU.

2. \( \theta = \{ Y=f(X), Z=a \} \) is an MGU.
   
   \[
   \begin{align*}
   p(f(X), Z) &\theta = p(f(X), a) \\
   p(Y, a) &\theta = p(f(X), a)
   \end{align*}
   \]

---

**MGU**

- The MGU is the "least specialized" way of making clauses with universal variables match.

- We can compute MGUs.

- Intuitively we line up the two formulas and find the first sub-expression where they disagree. The pair of subexpressions where they first disagree is called the disagreement set.

- The algorithm works by successively fixing disagreement sets until the two formulas become syntactically identical.

---

**MGU**

To find the MGU of two formulas \( f \) and \( g \).

1. \( k = 0; \sigma_0 = \{ \}; S_0 = \{ f, g \} \)
2. If \( S_k \) contains an identical pair of formulas stop, and return \( \sigma_k \) as the MGU of \( f \) and \( g \). 
3. Else find the disagreement set \( D_k = \{ e_1, e_2 \} \) of \( S_k \)
4. If \( e_1 = V \) a variable, and \( e_2 = t \) a term not containing \( V \) (or vice-versa) then let
   \[
   \begin{align*}
   \sigma_{k+1} &= \sigma_k \{ V=t \} \quad \text{(Compose** the additional substitution)} \\
   S_{k+1} &= S_k \{ V=t \} \quad \text{(Apply the additional substitution)} \\
   k &= k+1
   \end{align*}
   \]
5. Else stop, \( f \) and \( g \) cannot be unified.
   **Note that this is compose not conjoin!**
MGU Example 1

\[ S_0 = \{ p(f(a), g(X)) ; p(Y,Y) \} \]

MGU Example 2

\[ S_0 = \{ p(a,X,h(g(Z))) ; p(Z,h(Y),h(Y)) \} \]

MGU Example 3

\[ S_0 = \{ p(X,X) ; p(Y,f(Y)) \} \]

Non-Ground Resolution

- Resolution of non-ground clauses. From the two clauses
  \[(L, Q_1, Q_2, \ldots, Q_k) \]
  \[ (\neg M, R_1, R_2, \ldots, R_n) \]

  Where there exists \( \sigma \) a MGU for \( L \) and \( M \).

  We infer the new clause

  \[(Q_1 \sigma, \ldots, Q_k \sigma, R_1 \sigma, \ldots, R_n \sigma)\]
Non-Ground Resolution Example

1. \((p(X), q(g(X)))\)
2. \((r(a), q(Z), \neg p(a))\)
   \(L = p(X); M = p(a)\)
   \(\sigma = \{X = a\}\)
3. \(R[1a,2c]{X=a} (q(g(a)), r(a), q(Z))\)

The notation is important.
- "R" means resolution step.
- "1a" means the first (a-th) literal in the first clause i.e. \(p(X)\).
- "2c" means the third (c-th) literal in the second clause, \(\neg p(a)\).
- 1a and 2c are the "clashing" literals.
- \(\{X = a\}\) is the substitution applied to make the clashing literals identical.

Resolution Proof Example

"Some patients like all doctors. No patient likes any quack. Therefore no doctor is a quack."

Step 1: English -> FOL -> Clausal form

1A) Pick symbols to represent these assertions.
- \(p(X)\): X is a patient
- \(d(X)\): X is a doctor
- \(q(X)\): X is a quack
- \(l(X,Y)\): X likes Y

1B) Convert each assertion to a first-order formula.

Some patients like all doctors.

\[\exists X.(p(X) \land \forall Y.(d(Y) \rightarrow l(X,Y)))\] [F1]
Resolution Proof Example

Step 1: English -> FOL -> Clausal form
1B) Convert each assertion to a first-order formula.

Some patients like all doctors.
\[ \exists X. (p(X) \land \forall Y. (d(Y) \rightarrow l(X,Y))) \] [F1]

No patient likes any quack
????

Therefore no doctor is a quack.
???? [Query] 

Resolution Proof Example

Step 1: English -> FOL -> Clausal form
1B) Convert each assertion to a first-order formula.

Some patients like all doctors.
\[ \exists X. (p(X) \land \forall Y. (d(Y) \rightarrow l(X,Y))) \] [F1]

No patient likes any quack
\[ \forall X. (p(X) \rightarrow \forall Y. (q(Y) \rightarrow \neg l(X,Y))) \] [F2]

Therefore no doctor is a quack.

\[ \neg \exists X. (d(X) \land q(X)) \] [Query]

\[ \exists X. (d(X) \land q(X)) \] [Negated Query—for refutation proof] 

Conversion to Clausal Form

To convert the KB into Clausal form we perform the following 8-step procedure:

1. Eliminate Implications.
2. Move Negations inwards (and simplify \(\neg\neg\)).
3. Standardize Variables.
4. Skolemize.
5. Convert to Prenex Form.
6. Distribute disjunctions over conjunctions.
7. Flatten nested conjunctions and disjunctions.
8. Convert to Clauses.

Resolution Proof Example

Step 2. Resolution Refutation Proof from the clauses.
(try to derive \((\neg)\) – the empty clause)

1. \(p(a)\)
2. \((\neg d(Y), l(a,Y))\)
3. \((\neg p(Z), \neg q(R), \neg l(Z,R))\)
4. \(d(b)\)
5. \(q(b)\)
Answer Extraction

• The previous example shows how we can answer true-false questions. With a bit more effort we can also answer “fill-in-the-blanks” questions (e.g., what is wrong with the car?).

• For those who know Prolog, the strategy is similar. We use free variables in the query where we want to fill in the blanks. We simply need to keep track of the binding that these variables received in proving the query.

  • parent(art, jon) — is art one of jon’s parents?
  • parent(X, jon) — who is one of jon’s parents?

Answer Extraction: Example 1

1. father(art, jon)
2. father(bob,kim)
3. ( ¬father(Y,Z), parent(Y,Z)) i.e. all fathers are parents
4. ( ¬ parent(X,jon), answer(X)) i.e. the query is: who is parent of jon?

Here is a resolution proof:*

5. R[4,3b]{Y=X,Z=jon} ( ¬father(X,jon), answer(X))
6. R[5,1]{X=art} answer(art)
so art is parent of jon

* Remember a good strategy is to start a refutation proof by resolving the query since the source of the inconsistency you’re searching for is derived from the the negated query.

Answer Extraction: Example 2

1. (father(art, jon), father(bob,jon)) // either bob or art is parent of jon
2. father(bob,kim)
3. ( ¬father(Y,Z), parent(Y,Z)) // all fathers are parents
4. ( ¬ parent(X,jon), answer(X)) // query is parent(X,jon)

Here is a resolution proof:

5. R[4,3b]{Y=X,Z=jon} ( ¬father(X,jon), answer(X))
6. R[5,1]{X=art} (father(bob,jon), answer(art))
7. R[6,3a]{Y=bob,Z=jon} (parent(bob,jon), answer(art))
8. R[7,4]{X=bob} (answer(bob), answer(art))
A disjunctive answer: either bob or art is parent of jon.
### Factoring

1. \((p(X), p(Y))\) \hspace{1em} // i.e., \(\forall X, \forall Y \neg p(X) \rightarrow p(Y)\)
2. \((\neg p(V), \neg p(W))\) \hspace{1em} // i.e., \(\forall V, \forall W \ p(V) \rightarrow \neg p(W)\)

* These clauses are **intuitively contradictory**, but following the strict rules of resolution only we obtain:
3. \(R[1a,2a](X=V) \ (p(Y), \neg p(W))\)
   Renaming variables: \((p(Q), \neg p(Z))\)
4. \(R[3b,1a](X=Z) \ (p(Y), p(Q))\)

**Problem:** No way of generating empty clause!

**Important:** Factoring is needed to make resolution **complete**, without it resolution is incomplete!

---

### Factoring (dealing with duplicate literals in a clause)

If two or more literals from the **same** clause \(C\) have an MGU \(q\), then \(Cq\) with all duplicate literals removed is called a **factor** of \(C\).

Example:

\(C = (p(X), p(f(Y)), \neg q(X))\)
\(0 = \{X = f(Y)\}\)
\(C0 = (p(f(Y)), p(f(Y)), \neg q(f(Y)))\) so \((p(f(Y)), \neg q(f(Y)))\) is a factor of \(C\)

**Adding a factor of a clause can be a step of proof:**
1. \((p(X), p(Y))\) \hspace{1em} // potential duplicate literals
2. \((\neg p(V), \neg p(W))\) \hspace{1em} // potential duplicate literals
3. \(f[1ab](X=Y) p(Y)\) \hspace{1em} // clause 3 is a factor of clause 1
4. \(f[2ab](V=W) \neg p(W)\) \hspace{1em} // clause 4 is a factor of clause 2
5. \(R[3,4](Y=W) ()\) \hspace{1em} // resolve the factors to complete the proof

---

### Prolog*

**Prolog** search mechanism (without not and cut) is simply an instance of resolution, except

1. **Clauses are Horn** (only one positive literal)
2. **Prolog uses a specific depth first strategy when searching for a proof.** (Rules are used first mentioned first used, literals are resolved away left to right).

---

### Prolog* (An aside for those who know Prolog)

The Prolog Predicate "Append":
1. \(\text{append([], Z, Z)}\)
2. \(\text{append([E1 | R1], Y, [E1 | Rest]) :- \ append(R1, Y, Rest)}\)

**Note:**
- \(2\) is actually the clause \((\text{append([E1 | R1], Y, [E1 | Rest])} \ , \neg\text{append(R1,Y,Rest)})\)
- \([\]\) is a constant (the empty list)
- \([X | Y]\) is \(\text{cons}(X,Y)\).
- So \([a,b,c]\) is short hand for \(\text{cons}(a,\text{cons(b,cons(c,[\])))}\)

* You will not be tested on this material
### Prolog*: Example of proof

Try to prove:

\[
\text{append([a,b], [c,d], [a,b,c,d])}:
\]

1. \[\text{append([], Z, Z)}\]
2. \[\text{(append([E1|R1], Y, [E1|Rest]), } \neg \text{append(R1,Y,Rest))}\]
3. \[\neg \text{append([a,b], [c,d], [a,b,c,d])}\]
4. \[\text{R[3,2a][E1=a, R1=[b], Y=[c,d], Rest=[b,c,d]} \neg \text{append([b], [c,d], [b,c,d])}\]
5. \[\text{R[4,2a][E1=b, R1=[], Y=[c,d], Rest=[c,d]} \neg \text{append([], [c,d], [c,d])}\]
6. \[\text{R[5,1][Z=[c,d]]} ()\]

* You will not be tested on this material

### Review: One Last Example!

Consider the following English description

- Whoever can read is literate.
- Dolphins are not literate.
- Flipper is an intelligent dolphin.

- Who is intelligent but cannot read.

### Example:

Step 1: English -> FOL -> Clausal form
1. Pick symbols to represent these assertions.
2. Convert each assertion to a first-order formula.
3. Negate Query & Convert to Clausal Form (the 8 steps)

- Whoever can read is literate.
  \[\forall X. \text{read}(X) \rightarrow \text{lit}(X)\]
- Dolphins are not literate.
  \[\forall X. \text{dolp}(X) \rightarrow \neg \text{lit}(X)\]
- Flipper is an intelligent dolphin
  \[\text{dolp}(\text{flipper}) \land \text{intell} (\text{flipper})\]

- Who is intelligent but cannot read?
  \[\exists X. \text{intell}(X) \land \neg \text{read}(X)\]

### Example: convert to clausal form

Step 1: English -> FOL -> Clausal form
1A) Pick symbols to represent these assertions.
1B) Convert each assertion to a first-order formula.
1C) Negate Query & Convert to Clausal Form (the 8 steps)

- Whoever can read is literate.
  \[\forall X. \text{read}(X) \rightarrow \text{lit}(X)\]
  \[\neg \text{read}(X), \text{lit}(X)\]
- Dolphins are not literate.
  \[\forall X. \text{dolp}(X) \rightarrow \neg \text{lit}(X)\]
  \[\neg \text{dolp}(X), \neg \text{lit}(X)\]
- Flipper is an intelligent dolphin.
  \[\text{dolp}(\text{flipper}) \land \text{intell}(\text{flipper})\]
- Who is intelligent but cannot read?
  \[\exists X. \text{intell}(X) \land \neg \text{read}(X)\]
  \[\neg \text{intell}(X), \text{read}(X), \text{answer}(X)\]
Example: do the resolution proof

**Step 1: English to FO to Clausal Form**

1A) Pick symbols to represent these assertions.
1B) Convert each assertion to a first-order formula.
1C) Negate Query & Convert to Clausal Form (the 8 steps)
2) Do the resolution proof to derive ()

1. \(\neg \text{read}(X), \text{lit}(X)\)
2. \(\neg \text{dolp}(X), \neg \text{lit}(X)\)
3. \(\text{dolp}(\text{flip})\)
4. \(\text{intell}(\text{flip})\)
5. \(\neg \text{intell}(X), \text{read}(X), \text{answer}(X)\)
6. \(R[5a,4] X=\text{flip}. \ (\text{read}(\text{flip}), \text{answer}(\text{flip}))\)
7. \(R[6,1a] X=\text{flip}. \ (\text{lit}(\text{flip}), \text{answer}(\text{flip}))\)
8. \(R[7,2b] X=\text{flip}. \ (\neg \text{dolp}(\text{flip}), \text{answer}(\text{flip}))\)
9. \(R[8,3] \text{answer}(\text{flip})\)

so flip is intelligent but cannot read!

---

**KB—many models**

1. \(\text{on}(b,c)\)
2. \(\text{clear}(e)\)

---

**Review**

**Desired Properties of Reasoning**

**Soundness**

\[\text{If } \text{KB} \vdash \phi \text{ then } \text{KB} \models \phi\]

- i.e. all conclusions arrived at via the proof procedure are correct – they are logical consequences of the KB.

**Completeness**

\[\text{If } \text{KB} \models \phi \text{ then } \text{KB} \vdash \phi\]

- i.e. every logical consequence of the KB can be generated by the proof procedure.

Note proof procedures are computable, but they might have very high complexity in the worst case. So completeness is not necessarily achievable in practice.
Resolution Proof: Forward chaining

Logical consequences can be generated from the resolution rule in two ways:

1. Forward chaining inference ("Consequence Finding")
   - If we have a sequence of clauses C1, C2, ..., Ck
   - Such that each Cl is either in KB or is the result of a resolution step involving two prior clauses in the sequence.
   - We then have that KB ⊢ Ck.
   - Forward chaining is sound so we also have KB ⊨ Ck.

Resolution Proof: Refutation proofs

2. Refutation proofs
   - We determine if KB ⊢ f by showing that a contradiction can be generated from KB ∧ ¬f.
   - In this case a contradiction is an empty clause (1).
   - We employ resolution to construct a sequence of clauses C1, C2, ..., Cm such that
     - Ci is in KB ∧ ¬f, or is the result of resolving two previous clauses in the sequence.
     - Cm = () i.e. its the empty clause.

Conversion to Clausal Form

To convert the KB into Clausal form we perform the following 8-step procedure:

1. Eliminate Implications.
3. Standardize Variables.
4. Skolemize.
5. Convert to Prenex Form.
6. Distribute disjunctions over conjunctions.
7. Flatten nested conjunctions and disjunctions.
8. Convert to Clauses.

C-T-C-F: Skolemization Examples

• ∀XYZ ∃W.r(X,Y,Z,W) ⇒ ∀XYZ.r(X,Y,Z,h1(X,Y,Z))
• ∀XY∃W.r(X,Y,g(W)) ⇒ ∀XY.r(X,Y,g(h2(X,Y)))
• ∀XY∃W∀Z.r(X,Y,W) ∧ q(Z,W) ⇒ ∀XYZ.r(X,Y,h3(X,Y)) ∧ q(Z,h3(X,Y))