Knowledge Representation and Reasoning

Slides borrowed from Fahiem Bacchus, Sheila McIlraith, Ronald J. Brachman and Hector J. Levesque
Siri

Dialog driven, task-oriented, location-aware

- What knowledge does it represent?
- What reasoning must it do?
- Where does it fail?
- What would it take to extend it?
Consider the task of understanding a simple story
Consider the task of understanding a simple story
Three Little Pigs

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.
What is knowledge?

Or ... how can we talk about knowledge?

“John knows that ...” vs. “John fears that...”
- can be a true/false, right/wrong
- same content, different attitude,
  not necessarily true and/or held for appropriate reasons

Here, we don’t make a distinction, the main idea

Main idea → taking the world to be one way and not another
What is representation?

Symbols standing for things in the world

“John”

Knowledge Representation

→ symbolic encoding of propositions believed (by some agent)
Some difficult questions to answer...

Question Answering:
Q: Why do they have umbrellas?

Internal Textual Representation:
A group of people enjoying a sunny day at the beach with umbrellas in the sand.
Some difficult questions to answer...

Question Answering:
Q: Why do they have umbrellas?
A: Shade

Internal Textual Representation:
A group of people enjoying a sunny day at the beach with umbrellas in the sand.

External Knowledge:
An umbrella is a canopy designed to protect against rain or sunlight. Larger umbrellas are often used as points of shade on a sunny beach. A beach is a landform along the coast of an ocean. It usually consists of loose particles, such as sand....
KR and AI

- **KR&R started as a field in the context of AI research**
  - Need explicitly represented knowledge to achieve intelligent behaviour
    - Expert systems, language understanding
- **Many of AI problems today heavily rely on statistical representation and reasoning**
  - Speech understanding, vision, machine learning, natural language processing
- **Some AI problems require symbolic representation and reasoning**
  - Explanation, story generation
  - Planning, diagnosis
- **Many applications outside AI**
  - Bio-medicine, Engineering, Business and Commerce, Databases, Software Engineering, Education
"A how-to on manipulating the masses into unfaltering obedience."

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
  - Reasoning employed to make conclusions about given situations
  - More generally, reasoning provides an exponential or more compression in knowledge we need to store
    - i.e. without reasoning we would have to store an infeasible amount of information, including specifics, like elephants can’t fit into teacups
Utility of Logical Representations

- They are mathematically precise, thus we can analyze their limitations, properties, 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, they have well developed proof theories: formal procedures for reasoning at the syntactic level (achieved by manipulating sentences).
KR is first and foremost about knowledge

- Meaning and entailment
- Find individuals and properties, then encode facts sufficient for entailments
- Task: Knowledge base with appropriate entailments
  - What vocabulary?
  - What facts to represent?
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.

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

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.

CSC384 University of Toronto
Entailments

\[ \alpha \models \beta \]

- Read as “\( \alpha \) entails \( \beta \)” or “\( \beta \) follows logically from \( \alpha \)”
- Meaning, in any world in which \( \alpha \) is true, \( \beta \) is true also
- 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 \( \beta \)

➤ Example

\[ \neg (A \lor B) \vdash (\neg A \land \neg B) \]
First Order Logic

- Teaching CSC384, want to represent knowledge that would be useful for making the course a successful learning experience

- **Objects:**
  - Students, subjects, assignments, numbers

- **Relations:**
  - Difficult(subject), CSMajor(student), handedIn(student, assignment)

- **Functions**
  - Grade(student, assignment) $\rightarrow$ number
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
First Order Syntax

- Start with a set of primitive symbols
  1. Constant symbols
  2. Function symbols
  3. Predicate symbols
  4. Variables

Each function and predicate symbol has a specific *arity* (determines number of arguments it takes)
First Order Syntax

Atomic formula

\[ P(t_1, \ldots, t_k) \]

- Predicate
- Term
- Arity
Term

Terms are used as names (perhaps complex nested names) for objects in the domain.

Terms of the language can be:
- A constant → denote specific objects
- A variable → not yet determined, but will eventually denote particular objects
- An expression of the form $f(t_1, ..., t_k)$ → map tuples of objects to other objects
Term

Terms are used as names (perhaps complex nested names) for objects in the domain

Terms of the language can be:
- A constant → 5
- A variable → X, (where X = 5)
- An expression of the form \( f(t_1, \ldots, t_k) \) → \(+3, 2\)
Formula

\[ P(t_1, \ldots, t_k) \]

- **Terms** denote objects, **formulas** represent T/F assertions about these objects
- **Father_of(jane, bill)**, **female(jane)**, **system_down()**
First Order Syntax – Building Up

Consider a formula $f$:

The negation (NOT) of a formula is a new formula $
eg f$

Asserts that $f$ is false.
Consider a set of formulas $f_1, \ldots, f_n$:

- The conjunction (\textbf{AND}) of a set of formulas is a new formula

$$f_1 \land f_2 \land \ldots \land f_n$$

Asserts that each formula $f_i$ is true.
First Order Syntax – Building Up

Consider a set of formulas $f_1, \ldots, f_n$:

The disjunction (OR) of a set of formulas is a new formula

$$f_1 \lor f_2 \lor \ldots \lor f_n$$

Asserts that at least one formula $f_i$ is true.
First Order Syntax – Building Up

Consider a formula $f$:

**Existential Quantification**

$$\exists X. f \text{ (where } X \text{ is a variable)}$$

Asserts there is some individual such that $f$ under that binding will be true
First Order Syntax – Building Up

Consider a formula $f$:

- **Universal Quantification**
  
  $\forall X. f$ (where $X$ is a variable)

  Asserts that $f$ is true for every individual.
First Order Syntax - Abbreviations

Implication ($\rightarrow$)

$f_1 \rightarrow f_2$ 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 that 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 recursive structure of syntax, so we can give any formula, no matter how complex a mapping to semantic entities
Semantics

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 \( L(F, P, V) \)

- \( F = \) set of function (and constant symbols)
  - Each symbol \( f \) in \( F \) has a particular arity
- \( P = \) set of predicate and relation symbols
  - Each symbol \( p \) in \( P \) has a particular arity
- \( V = \) an infinite set of variables
Model

Domain  
(non-empty set of individuals)

$\langle D, \Phi, \Psi, \nu \rangle$

Variable assignment  
Maps every variable to some individual

Function mapping  
$\Phi(f) \to (D^k \to D)$

Predicate Mapping  
$\Psi(p) \to (D^k \to \text{T/F})$
Model

Domain
(non-empty set of individuals) \( \langle D, \Phi, \Psi, v \rangle \)

- \( d \in D \) is an individual
  - E.g. \{Michael, Rome, GrandHotel, 3stars\}
- Domains often infinite, but we’ll use finite models to prime our intuitions
Model

Function mapping
\( \Phi(f) \to (D^k \to D) \)

- Given a k-ary function \( f \), over \( k \) individuals
  - What individual does \( f(d_1, ..., d_k) \) denote?
- 0-ary functions (constants) mapped to specific individuals
  - \( \Phi(\text{client17}) = \text{Michael}, \Phi(\text{hotel5}) = \text{grandhotel} \)
- 1-ary functions are mapped to functions in \( D \to D \)
  - \( \Phi(\text{minquality}) = f_{\text{minquality}}: f_{\text{minquality}}(\text{Michael}) = 3\text{stars} \)
  - \( \Phi(\text{rating}) = f_{\text{rating}}: f_{\text{rating}}(\text{grandhotel}) = 5\text{stars} \)
- N-ary functions are mapped similarly
Predicate Mapping

\[ \Psi(p) \rightarrow (D^k \rightarrow \text{T/F}) \]

\[ \langle D, \Phi, \Psi, v \rangle \]

- **Given a k-ary predicate** \( p \), **over k individuals**
  - Does \( p(d_1, ..., d_k) \) hold true?

- 0-ary functions (constants) mapped to specific individuals
  - \( \Psi(\text{rainy}) = \text{True} \)
  - \( \Psi(\text{sunny}) = \text{False} \)

- 1-ary predicates are mapped indicator functions of subsets of \( D \)
  - \( \Psi(\text{satisfied}) = p_{\text{satisfied}}: p_{\text{satisfied}}(\text{Michael}) = \text{True} \)
  - \( \Psi(\text{privatebeach}) = p_{\text{privatebeach}}: p_{\text{privatebeach}}(\text{grandhotel}) = \text{False} \)

- N-ary predicates map similarly
Model

Variable assignment
Maps every variable to some individual

\[ \langle D, \Phi, \Psi, v \rangle \]

- 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.
Models—Examples.

Environment

Language (Syntax)

Constants: a, b, c, e

Functions:
No function

Predicates:
- on: binary
- above: binary
- clear: unary
- ontable: unary
Models—Examples.

Language (syntax)
- Constants: a, b, c, e
- Predicates:
  - on (binary)
  - above (binary)
  - clear (unary)
  - ontable (unary)

A possible Model $I_1$ (semantics)
- $D = \{A, B, C, E\}$
- $\Phi(a) = A$, $\Phi(b) = B$, $\Phi(c) = C$, $\Phi(e) = E$.
- $\Psi(\text{on}) = \{(A, B), (B, C)\}$
- $\Psi(\text{above}) = \{(A, B), (B, C), (A, C)\}$
- $\Psi(\text{clear}) = \{A, E\}$
- $\Psi(\text{ontable}) = \{C, E\}$
Models—Examples.

Model \( I_1 \)

\[
D = \{A, B, C, E\}
\]

\[
\Phi(a) = A, \ \Phi(b) = B, \ \Phi(c) = C, \ \Phi(e) = E.
\]

\[
\Psi(\text{on}) = \{(A,B),(B,C)\}
\]

\[
\Psi(\text{above}) = \{(A,B),(B,C),(A,C)\}
\]

\[
\Psi(\text{clear}) = \{A,E\}
\]

\[
\Psi(\text{ontable}) = \{C,E\}
\]
Models—Formulas true or false?

Model $I_1$

$D = \{A, B, C, E\}$

$\Phi(a) = A, \Phi(b) = B, \Phi(c) = C, \Phi(e) = E.$

$\Psi(\text{on}) = \{(A,B),(B,C)\}$

$\Psi(\text{above}) = \{(A,B),(B,C),(A,C)\}$

$\Psi(\text{clear}) = \{A,E\}$

$\Psi(\text{ontable}) = \{C,E\}$

$\forall X, Y. \ on(X,Y) \rightarrow \text{above}(X,Y)$

$X=A, Y=B$

$X=C, Y=A$

$X=A, Y=C$

$\forall X, Y. \ \text{above}(X,Y) \rightarrow \text{on}(X,Y)$

$X=A, Y=B$

$X=A, Y=C$
Models—Examples.

Model \( M_1 \)

\[
\begin{align*}
D &= \{A, B, C, E\} \\
\Phi(a) &= A, \quad \Phi(b) = B, \quad \Phi(c) = C, \quad \Phi(e) = E. \\
\Psi(on) &= \{(A, B), (B, C)\} \\
\Psi(above) &= \{(A, B), (B, C), (A, C)\} \\
\Psi(clear) &= \{A, E\} \\
\Psi(ontable) &= \{C, E\}
\end{align*}
\]

\( \forall X \exists Y. \left( \text{clear}(X) \lor \text{on}(Y, X) \right) \)

- \( X = A \)  
- \( X = C, \ Y = B \)  
- \( \ldots \)

\( \exists Y \forall X. \left( \text{clear}(X) \lor \text{on}(Y, X) \right) \)

- \( Y = A \)  ?  No!  (\( X = C \))
- \( Y = C \)  ?  No!  (\( X = B \))
- \( Y = E \)  ?  No!  (\( X = B \))
- \( Y = B \)  ?  No!  (\( X = B \))
KB—many models

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 intend that the real world (i.e. our set theoretic abstraction of it) is one of its models.
- This means that every statement in KB is true in the real world.
- Note however, that not everything true in the real 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.,
  \[ I \models KB \rightarrow I \models f. \]

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

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

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

---

??? If \( KB \) doesn’t entail \( f \), is \( f \) false in the real world?
Logical Consequence Example

- **elephant(clyde)**
  - the individual denoted by the symbol *clyde* in the set denoted by *elephant* (has the property that it is an *elephant*).

- **teacup(cup)**
  - *cup* is a teacup.

Note that in both cases a unary predicate specifies a set of individuals. Asserting a unary predicate to be true of a term means that the individual denoted by that term is in the specified set.

- Formally, we map individuals to TRUE/FALSE (this is an indicator function for the set).
Logical Consequence Example

For all pairs of individuals if the first is an elephant and the second is a teacup, then the pair of objects are related to each other by the largerThan relation.
Logical Consequence Example

\( \forall X,Y. \text{largerThan}(X,Y) \rightarrow \neg \text{fitsIn}(X,Y) \)

- For all pairs of individuals if \( X \) is larger than \( Y \) (the pair is in the largerThan relation) then we cannot have that \( X \) fits in \( Y \) (the pair cannot be in the fitsIn relation).

- (The relation largerThan has a empty intersection with the fitsIn relation).
Logical Consequences

- \( \neg \text{fitsIn}(\text{clyde},\text{cup}) \)

- We know \( \text{largerThan}(\text{clyde},\text{teacup}) \) from the first implication. Thus we know this from the second implication.
Logical Consequences

fitsIn

¬fitsIn

Elephants $\times$ teacups
(clyde, cup)
Logical Consequence Example

- If an interpretation satisfies KB, then the set of pairs *elephant X teacup* must be a subset of *largerThan*, which is disjoint from *fitsIn*.

- Therefore, the pair *(clyde,cup)* must be in the complement of the set *fitsIn*.

- Hence, ¬*fitsIn*(clyde,cup) must be true in every interpretation that satisfies KB.

- ¬*fitsIn*(clyde,cup) is a logical consequence of KB.
Models and Interpretations

- 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 the “real world”! Because Each sentence in KB rules out certain unintended interpretations.

- This is called axiomatizing the domain
Computing logical consequences

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

- This would allow us to reason with our knowledge
  - Represent the knowledge as logical formulas
  - Apply procedures for generating logical consequences

- These procedures are called proof procedures.
Proof Procedures

- Interesting, 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 develop a proof procedure for first-order logic called resolution.
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 $\text{KB} \vdash f$ to indicate that $f$ can be proved from $\text{KB}$ (the proof procedure used is implicit).
Properties of Proof Procedures

- **Soundness**
  - $KB \not\vdash f \rightarrow KB \not\models f$
  - i.e. all conclusions arrived at via the proof procedure are correct: they are logical consequences.

- **Completeness**
  - $KB \models f \rightarrow KB \not\vdash f$
  - 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

- **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 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))
    \[\Rightarrow \neg\text{largerThan(clyde,cup)}\]
Resolution Proof: Forward chaining

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

1. Forward Chaining inference.
   - If we have a sequence of clauses $C_1, C_2, \ldots, C_k$
   - Such that each $C_i$ is either in $KB$ or is the result of a resolution step involving two prior clauses in the sequence.
   - We then have that $KB \vdash C_k$.

Forward chaining is sound so we also have $KB \models C_k$
Resolution Proof: Refutation proofs

2. Refutation proofs.
   ▶ We determine if $KB \vdash f$ by showing that a contradiction can be generated from $KB \land \neg f$.
   ▶ In this case a contradiction is an empty clause ($\bot$).
   ▶ We employ resolution to construct a sequence of clauses $C_1, C_2, \ldots, C_m$ such that
     ▶ $C_i$ is in $KB \land \neg f$, or is the result of resolving two previous clauses in the sequence.
     ▶ $C_m = \bot$, i.e. its the empty clause.
Resolution Proof: Refutation proofs

- If we can find such a sequence $C_1, C_2, \ldots, C_m=()$, we have that $\text{KB} \vdash f$.
- Furthermore, this procedure is sound so $\text{KB} \models f$.
- And the procedure is also complete so it is capable of finding a proof of any $f$ that is a logical consequence of $\text{KB}$. I.e. $\text{If } \text{KB} \models f \text{ then we can generate a refutation from } \text{KB} \land \neg f$. 
Resolution Proofs Example

Want to prove \( \text{likes}(clyde, peanuts) \) from:

1. \((\text{elephant}(clyde), \text{giraffe}(clyde))\)
2. \((\neg \text{elephant}(clyde), \text{likes}(clyde, peanuts))\)
3. \((\neg \text{giraffe}(clyde), \text{likes}(clyde, leaves))\)
4. \(\neg \text{likes}(clyde, leaves)\)

Forward Chaining Proof:

- \[3\&4 \rightarrow \neg \text{giraffe}(clyde) \text{ [5.]}\]
- \[5\&1 \rightarrow \text{elephant}(clyde) \text{ [6.]}\]
- \[6\&2 \rightarrow \text{likes}(clyde, peanuts) \text{ [7.]} \checkmark\]
Resolution Proofs Example

1. (elephant(clyde), giraffe(clyde))
2. (¬elephant(clyde), likes(clyde,peanuts))
3. (¬giraffe(clyde), likes(clyde,leaves))
4. ¬likes(clyde,leaves)

Refutation Proof:
- ¬likes(clyde,peanuts) [5.]
- 5&2 → ¬elephant(clyde) [6.]
- 6&1 → giraffe(clyde) [7.]
- 7&3 → likes(clyde,leaves) [8.]
- 8&4 → () √
Resolution Proofs

- Proofs by refutation have the advantage that they are easier to find.
  - They are more focused to the particular conclusion we are trying to reach.

- To develop a complete 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 (unification).
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 Prenix Form.
6. Distribute conjunctions over disjunctions.
7. Flatten nested conjunctions and disjunctions.
8. Convert to Clauses.
We use this example to show each step:
$$\forall X. p(X) \rightarrow ((\forall Y. p(Y) \rightarrow p(f(X,Y))) \land \neg(\forall Y. \neg q(X,Y) \land p(Y)))$$

1. Eliminate implications: $$A \rightarrow B \Rightarrow \neg A \lor B$$

$$\forall X. \neg p(X)$$
$$\lor (\forall Y. \neg p(Y) \lor p(f(X,Y))) \land \neg(\forall Y. \neg q(X,Y) \land p(Y))$$
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 \( \neg \neg \))

\[ \forall X. \neg p(X) \lor (\forall Y. \neg p(Y) \lor p(f(X,Y))) \land (\exists Y. q(X,Y) \lor \neg p(Y)) \]
Rules for moving negations inwards

- \( \neg (A \land B) \Rightarrow \neg A \lor \neg B \)
- \( \neg (A \lor B) \Rightarrow \neg A \land \neg B \)
- \( \neg \forall X. f \Rightarrow \exists X. \neg f \)
- \( \neg \exists X. f \Rightarrow \forall X. \neg f \)
- \( \neg \neg A \Rightarrow A \)
C-T-C-F: Standardize Variables

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

3. Standardize Variables (Rename variables so that each quantified variable is unique)

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

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 $a$. 
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 original quantified formula we know nothing about the variable “Y”. Just what was being asserted by the existential formula.
C-T-C-F: Skolemization

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.
C-T-C-F: Skolemization

- 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

\[ \forall XYZ \exists W. r(X,Y,Z,W) \rightarrow \forall XYZ. r(X,Y,Z,h_1(X,Y,Z)) \]

\[ \forall XY \exists W. r(X,Y,g(W)) \rightarrow \forall XY. r(X,Y,Z,g(h_2(X,Y))) \]

\[ \forall XY \exists W. \forall Z. r(X,Y,W) \land q(Z,W) \rightarrow \forall XYZ. r(X,Y,h_3(X,Y)) \land q(Z,h_3(X,Y)) \]
C-T-C-F: Convert to prenex

∀X. ¬p(X)

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

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

∀X∀Y. ¬p(X)

\[ \bigvee \left( \neg p(Y) \vee p(f(X,Y)) \land q(X,g(X)) \vee \neg p(g(X)) \right) \]
∀X ∀Y. ¬p(X) 

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

6. Conjunctions over disjunctions

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

∀XY. (¬p(X) ∨ ¬p(Y) ∨ p(f(X,Y))) ∧ (¬p(X) ∨ q(X,g(X)) ∨ ¬p(g(X)))
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 (remove 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) \(\neg p(X) \lor \neg p(Y) \lor p(f(X,Y))\)
   b) \(\neg p(X) \lor q(X,g(X)) \lor \neg p(g(X))\)