1

ALGORITHMIC ABSTRACTIONS OF (LOGICAL) INFERENCE

Vasant Honavar

Artificial Intelligence Research Laboratory
College of Information Sciences and Technology
Pennsylvania State University

1 INTRODUCTION TO KNOWLEDGE REPRESENTATION

Logic is a way of representing knowledge. The role of Knowledge Representation in AI is primarily to reduce problems requiring intelligence to search problems.

Knowledge represenation is about *describing* objects, events, relationships, etc. in some domain of interest. The ability to describe the world assumes the existence of a language with appropriate structure (syntax) and meaning (semantics).

In what follows, we will quickly review propositional logic (Boolean logic) which should be familiar to most of the readers. In the propositional language, we have a countably infinite set of atoms, and two distinguished atoms (True, False) and logical connectives $\vee$, $\wedge$ $\neg$, $\rightarrow$, etc. A well formed sentence in propositional logic could be an atom, or a sentence that is obtained by using atom(s) and connectives according to certain syntactic rules. Thus, if ω is a sentence, so is $\neg\omega$. If ω_1 and ω_2 are sentences, then so are $\omega_1 \vee \omega_2$, $\omega_1 \wedge \omega_2$, and $\omega_1 \rightarrow \omega_2$. Atoms and their negations are called *literals*. It is common to use *extra-linguistic* symbols such as parentheses to group sentences.

Semantics has to do with associating elements of the logical language with the properties of the domain of discourse. For instance, we might use the logical proposition B to *denote* the fact that *the battery is charged*. It is important to emphasize that the atoms do not have any intrinsic meaning. An association of atoms with propositions about the world is called an *interpretation*.

In a given interpretation, the proposition associated with an atom is called its *denotation*. Under a *chosen* interpretation (i.e., association of atoms with propositions about the world), atoms have truth values True or False. Thus, we will assign binary truth values to atoms, yielding a two-valued logic (where sentences are either True or False with respect to a given interpretation). It is important to emphasize that True and False have no intrinsic meaning. This becomes clear when we consider the formal notion of semantics for propositional logic.

For now, suppose we have a language with no logical symbols. Thus, all we have are the atomic sentences. Let the atomic sentences be Rich, Poor.

Def: A *model* M is a subset of the set A of atomic sentences in our language.

By a model M $\subseteq$ A we will mean the state of affairs in which every atomic sentence in M is true, and every sentence not in M is false. (Note: "True" and "False" have no intrinsic meaning!)

The possible models in our example are:

M_0	{ }
M_1	{ Rich}
M_2	{ Poor}
M_3	{ Rich, Poor}

Models can be thought of as "possible worlds".

Rich	is true in	M_2, M_3
Rich $\vee$ Poor	is true in	M_1, M_2, M_3
Rich $\wedge$ Poor	is true in	M_3
Rich $\Rightarrow \neg$Poor	is true in	M_0, M_1, M_2
$\neg$Rich $\vee \neg$Poor	is true in	M_0, M_1, M_2

Def: Given two sentences, p and q we say that p *entails* q (written as p $\models$ q) if q holds (q is true) in every model in which p holds (See figure **??**).

For example, $p \wedge (p \Rightarrow q) \models q$

$$p \in M \text{ and } \neg p \vee q \in M$$

$$p \in M \wedge \neg p \vee q \in M$$

$$(p \wedge \neg p) \vee (p \wedge q) \in M$$

$$p \wedge q \in M$$

The relevance of predicate logic to AI hinges on the $\models$ realation.

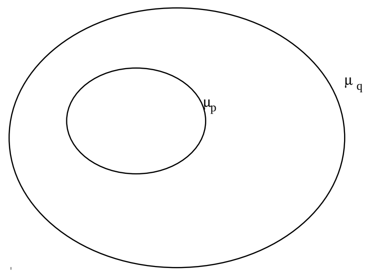

Figure 1 p $\models$ q, μ_q is the set of models in which q holds

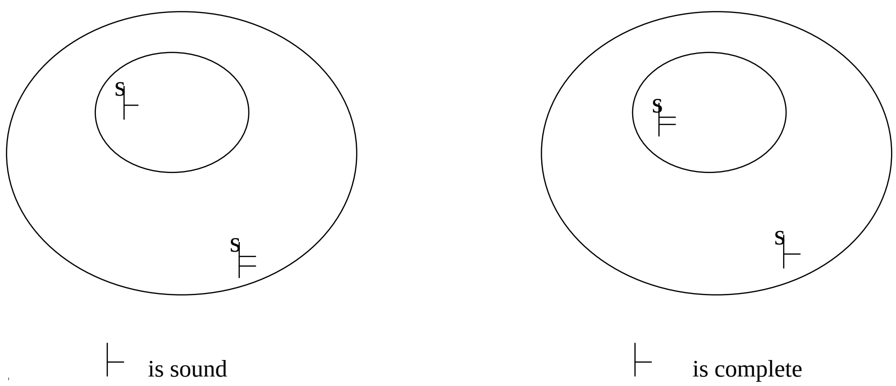

Figure 2 Soundness and Completeness of $\vdash$

Suppose an agent believes in $p_1, \ldots p_m$, then one can argue that he/she should be justified in concluding q wherever $p_1 \wedge \ldots \wedge p_m \models q$.

In general, trying to determine if p $\models$ q by enumerating the models in μ_q and μ_p and verifying $\mu_q \subseteq \mu_p$ is not feasible.

An alternative approach is to come up with an inference rule/procedure that derives a sentence q (of a prescribed syntactic form) wherever a sentence p (of a prescribed syntactic form) is given.

$$\left.\begin{array}{c} p \Rightarrow q \\ p \\ \hline q \end{array}\right\} \quad p \wedge (p \Rightarrow q)$$

$$\frac{\text{day\_of\_the\_week}(friday)}{q}$$

Since the inference may sanction inferences that may or may not be sanctioned by $\models$, we should distinguish the inference procedures used from $\models$.

We will say that $p \vdash_a q$ if q is *derivable* from p using a rule or set of rules or a procedure a.

Let $S_{\models}$ be the set of sentences sanctioned by $\models$.
Let $S_{\vdash}$ be the set of sentences deriviable using $\vdash$.

If $\vdash_a$ allows you to derive only those sentences sanctioned by $\models$, then we say that a is *sound*. (See Figure **??**)

If $\vdash_a$ allows you to derive all sentences sanctioned by $\models$, we say that $\vdash_a$ is *complete*.(See Figure **??**)

Ideally, we want inference procedures that are both sound and complete.

First Order Predicate Logic(FOPL) extends propositionsal logic in the following manner:

- It provides "quantifiers" that allow us to talk about all or some objects in our domain. E.g. $\forall x$ apple(x) $\Rightarrow$ sweet(x), or loves(John, *dog_of*(John)).

1.1 FOPL Syntax

Logical Symbols(Connectives)
NOT $\neg$ OR $\vee$ AND $\wedge$ IMPLIES $\Rightarrow$ EQUIVALENCE $\Leftrightarrow$

Quantifiers FORALL $\forall$ THERE EXISTS $\exists$

Non-Logical Symbols constants, or an infinite set of variables (e.g., x, y, ...)

Function Symbols e.g., $func1(x)$

Predicate Symbols e.g., apple(x), (i.e. those that have truth values associated with them)

Sentences

atomic sentences, (or literals) e.g. apple(x)

compound sentences, e.g. $\forall x$ apple(x) $\Rightarrow$ sweet(x), or $\exists x$ big(x)$\wedge$ house(x).

Terms constants, variables, or functional expressions ($f(x_1, \cdots x_n)$)

Functional expressions can be used instead of variables Example

Given the following predicates: purple(x) mushroom(x) poisonous(x) equal(x, y)

Express the following sentences in FOPL:

1. All purple mushrooms are poisonous.
   $\forall x$ [[purple(x) $\wedge$ mushroom(x)] $\Rightarrow$ poisonous(x)]
2. No purple mushroom is poisonous.
   $\forall x$ mushroom(x) $\wedge$ purple(x) $\Rightarrow \neg$ poisonous(x)
3. There is exactly one mushroom.
   [$\exists x$ mushroom(x) $\wedge$ $\forall y$ mushroom(y) $\Rightarrow$ equal(x, y)]

1.2 Semantics

Suppose we have a language with no logical symbols or variables (we have only predicate symbols and constants). E.g., predicates: Rich, Poor and constants: Tom

We have the atomic sentences: { Rich(Tom) and Poor(Tom) }.

Def: A *model* M is a subset of the set A of atomic sentences in our language.

By a model M $\subseteq$ A we will mean the state of affairs in which every atomic sentence in M is true, and every sentence not in M is false. (Note: "True" and "False" have no intrinsic meaning!)

Back to our example: The possible models are:

M_0	{ }
M_1	{ Rich(Tom)}
M_2	{ Poor(Tom)}
M_3	{ Rich(Tom), Poor(Tom)}

Models can be thought of as "possible worlds".

Rich(Tom)	is true in	M_2, M_3
Rich(Tom) $\vee$ Poor(Tom)	is true in	M_1, M_2, M_3
Rich(Tom) $\wedge$ Poor(Tom)	is true in	M_3
Rich(Tom) $\Rightarrow \neg$Poor(Tom)	is true in	M_0, M_1, M_2
$\neg$Rich(Tom) $\vee$ $\neg$Poor(Tom)	is true in	M_0, M_1, M_2

Consider the FOPL system with $P(x)$ and $D_x = \{a, b, c\}$ (where D_x is the domain for x). Then, the set of all possible models is
$M = \{P(a), P(b), P(c), P(a) \wedge P(b), P(a) \wedge P(c), P(b) \wedge P(c), P(a) \wedge P(b) \wedge P(c)\}$

Quantifiers

$\forall x\ P(x)$ is the same as $P(a_1) \wedge P(a_2) \wedge P(a_2) \wedge \ldots \wedge P(a_n)$

Suppose $D_x = \{a_1, a_2\}$, then $\forall x P(x)$ is true in any model that contains $P(a_1)$ and $P(a_2)$.

$\exists x P(x)$ is an infinite version of $\vee$. $\exists x P(x)$ is true in any model that contains at least one of $P(a_1)$, $P(a_2)$.

Example:
Consider a FOPL system with predicates: $P(x)$, $Q(x, y)$
Dx = {a, b} (domain of x)
Dy = {b, c} (domain of y)

1. Enumerate the set of models.
2. Identify the model(s) in which $\exists x \exists y Q(x, y)$ holds.
3. Identify the models in which $\forall x P(x)$ holds.
4. Identify the models in which $\neg\ \forall x P(x)$ holds.

<u>Anwser</u>:

1.
$M_0 = \{ \}$
$M_1 = \{ P(a) \}$
$M_2 = \{ P(b) \}$
$M_3 = \{ P(a)\ P(b) \}$
$M_4 = \{ Q(a,b) \}$
$M_5 = \{ Q(b,b) \}$
$M_6 = \{ Q(a,c) \}$
$M_7 = \{ Q(b,c) \}$
$M_8 = \{ P(a)\ Q(a,b) \}$
$M_9 = \{ P(a)\ Q(b,b) \}$
$M_10 = \{ P(a)\ Q(a,c) \}$
$M_11 = \{ P(a)\ Q(b,c) \}$
$M_12 = \{ P(b)\ Q(a,b) \}$
$M_13 = \{ P(b)\ Q(b,b) \}$
$M_14 = \{ P(b)\ Q(a,c) \}$
$M_15 = \{ P(b)\ Q(b,c) \}$
$M_16 = \{ P(a)\ P(b)\ Q(a,b) \}$
$M_17 = \{ P(a)\ P(b)\ Q(b,b) \}$
$M_18 = \{ P(a)\ P(b)\ Q(a,c) \}$
$M_19 = \{ P(a)\ P(b)\ Q(b,c) \}$

2. M4, M5, M6, M7, M8, M9, M10, M11, M12, M13, M14, M15, M16, M17, M18, and M19.

3. M3, M16, M17, M18, and M19.

4. M1, M2, M8, M9, M10, M11, M12, M13, M14, and M15.

1.3 Entailment and Inference

We can define the notions of entailment, and soundness, and completeness of inference rules for predicate logic in a manner analogous to propositional logic.

First, we consider the case without variables (and hence without quantifiers). Then we generalize.

Modus ponens:

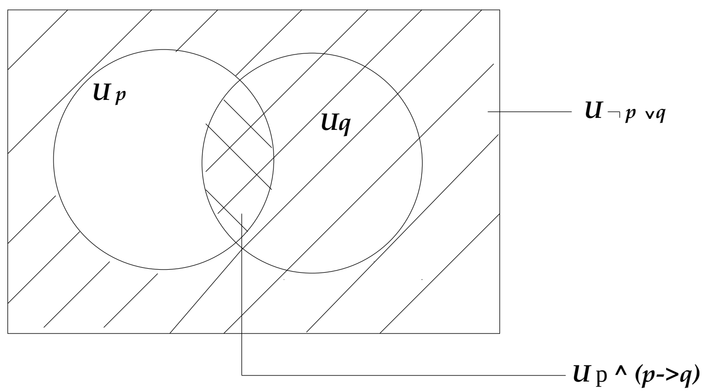

Figure 3 Illustration that $\mu_p \wedge$ (p $\rightarrow$ q) $\subseteq \mu_q$.

$$\left.\begin{array}{l} p \Rightarrow q \\ \underline{p \qquad\quad} \end{array}\right\} \; p \wedge (p \Rightarrow q)$$
$$\quad q$$

p
p $\Rightarrow$ q
these two conditions are given. q is infered.

We will show that Modus ponens is sound.

(p $\Rightarrow$ q) $\equiv$ ¬ p $\vee$ q

$\mu_p \wedge$ (p $\Rightarrow$ q) $\subseteq \mu_q$

so, p $\wedge$ (p $\Rightarrow$ q) $\models$ q.

therefore, Modus ponens is sound.

Ideally, we want inference procedures that are both sound and complete. Is MP complete? We will show that MP is not complete using proof by counter example.

Proof (by counter example)

```
cs472 meets at 1pm.
classes at ISU meet on MWF or TR.
Joe has to work at 1pm on RF.
Can Joe take cs472?
```

We will represent these facts in logic:

$$
\begin{aligned}
&T \Rightarrow \mathrm{tr}(cs472, 1pm) \vee \mathrm{mwf}(cs472, 1pm)\\
&\mathrm{tr}(cs472, 1pm) \wedge \mathrm{busy}(t, 1pm) \Rightarrow \mathrm{conflict}(cs472)\\
&\mathrm{tr}(cs472, 1pm) \wedge \mathrm{busy}(r, 1pm) \Rightarrow \mathrm{conflict}(cs472)\\
&\mathrm{mwf}(cs472, 1pm) \wedge \mathrm{busy}(m, 1pm) \Rightarrow \mathrm{conflict}(cs472)\\
&\mathrm{mwf}(cs472, 1pm) \wedge \mathrm{busy}(w, 1pm) \Rightarrow \mathrm{conflict}(cs472)\\
&\mathrm{mwf}(cs472, 1pm) \wedge \mathrm{busy}(f, 1pm) \Rightarrow \mathrm{conflict}(cs472)\\
&T \Rightarrow \mathrm{busy}(r, 1pm)\\
&T \Rightarrow \mathrm{busy}(f, 1pm)
\end{aligned}
$$

Goal: To prove that "conflict(cs472)". (It cannot be done with MP) We cannot show that conflict(cs472) holds given the axioms using MP alone. This is a situation where something can be shown to be correct but cannot be derived just based on the rule.

So, we need to modify MP so that we can have a complete inference rule that is also sound.

1.4 Toward a sound and complete inference rule

$$
\frac{\begin{array}{c} p \Rightarrow q \\ p \end{array}}{q}
$$

This is sound.

Since p doesn't have to be an atomic sentence, we can rewrite this inference rule in a more general form:

$$\frac{\begin{array}{c} a_1 \wedge a_2 \wedge a_3 \cdots a_{i-1} \wedge a_i \wedge a_{i+1} \cdots a_n \Rightarrow b \\ T \Rightarrow a_i \end{array}}{a_1 \wedge a_2 \wedge a_3 \cdots a_{i-1} \wedge a_{i+1} \cdots a_n \Rightarrow b}$$

It is easy to show that the above rule is sound.

In the following, assume $c = a_i$:

$$\frac{\begin{array}{c} a_1 \wedge a_2 \wedge a_3 \cdots a_{i-1} \wedge a_i \wedge a_{i+1} \cdots a_n \Rightarrow b \\ d_1 \wedge d_2 \cdots d_m \Rightarrow c \end{array}}{a_1 \wedge a_2 \wedge a_3 \cdots a_{i-1} \wedge a_{i+1} \cdots a_n \wedge d_1 \wedge d_2 \cdots d_m \Rightarrow b}$$

Sentences of the form $a_1 \wedge a_2 \wedge \cdots a_n \Rightarrow b$ are called *Horn Clauses*, and it can be shown that the inference rule is sound and complete for *Horn Clauses*. What if the sentences we have to deal with are not Horn Clauses?

<u>Theorem</u> M.P. is not complete for sentences that contain disjunctions.

We will extend MP to obtain an inference rule that is both sound and complete.

First of all, b doesn't have to be atomic, too. So, we can have the following sentences:
$a_1 \wedge a_2 \wedge \cdots a_{i-1} \wedge a_i \wedge a_{i+1} \wedge \cdots a_n \Rightarrow b_1 \vee b_2 \vee \cdots \vee b_k$
$d_1 \wedge d_2 \cdots d_m \Rightarrow c$ (assume $a_i = c$)

$(a_1 \wedge a_2 \cdots a_{i-1} \wedge a_{i+1} \cdots \wedge a_n) \wedge (d_1 \wedge d_2 \cdots \wedge d_m) \Rightarrow b_1 \vee b_2 \cdots \vee b_k$

As before, this rule can be shown to be sound.

$a_1 \wedge a_2 \wedge \cdots a_{i-1} \wedge a_i \wedge a_{i+1} \wedge \cdots a_n \Rightarrow b_1 \vee b_2 \vee \cdots \vee b_k$
$d_1 \wedge d_2 \cdots d_m \Rightarrow c_1 \vee c_2 \vee \cdots c_{j-1} \vee c_j \vee c_{j+1} \vee \cdots c_l$

$(a_1 \wedge a_2 \cdots a_{i-1} \wedge a_{i+1} \cdots \wedge a_n) \wedge (d_1 \wedge \cdots \wedge d_m) \Rightarrow$
$(b_1 \vee b_2 \cdots \vee b_k) \vee (c_1 \vee c_2 \vee \cdots c_{j-1} \vee c_{j+1} \vee \cdots c_l)$ (assume $c_j = a_i$)

This is the so-called *resolution principle* which is sound and complete for propositional logic. This rule is used to show that conflict(cs472) can be derived from the axioms.

1.5 Example

To show that conflict(cs472) can be derived from the following axioms using resolution principle.

$$T \to tr(cs472, 1pm) \quad \vee \quad mwf(cs472, 1pm) \tag{1.1}$$
$$busy(m, 1pm) \wedge mwf(cs472, 1pm) \implies conflict(cs472) \tag{1.2}$$
$$busy(w, 1pm) \wedge mwf(cs472, 1pm) \implies conflict(cs472) \tag{1.3}$$
$$busy(f, 1pm) \wedge mwf(cs472, 1pm) \implies conflict(cs472) \tag{1.4}$$
$$busy(t, 1pm) \wedge tr(cs472, 1pm) \implies conflict(cs472) \tag{1.5}$$
$$busy(r, 1pm) \wedge tr(cs472, 1pm) \implies conflict(cs472) \tag{1.6}$$
$$T \to busy(r, 1pm) \tag{1.7}$$
$$T \to busy(f, 1pm) \tag{1.8}$$

Goal: to show conflict(cs472).

Using equations (11.4) and (11.8)

$$\begin{aligned} busy(f, 1pm) \wedge mwf(cs472, 1pm) &\implies conflict(cs472) \\ T &\to busy(f, 1pm) \end{aligned}$$

$$mwf(cs472, 1pm) \implies conflict(cs472) \tag{1.9}$$

Using equations (11.6) and (11.7)

$$\begin{aligned} busy(r, 1pm) \wedge tr(cs472, 1pm) &\implies conflict(cs472) \\ T &\to busy(r, 1pm) \end{aligned}$$

$$tr(cs472, 1pm) \implies conflict(cs472) \tag{1.10}$$

Using equations (11.10) and (11.1)

$$\begin{aligned} tr(cs472, 1pm) &\implies conflict(cs472) \\ T &\to tr(cs472, 1pm) \vee mwf(cs472, 1pm) \end{aligned}$$

$$T \rightarrow conflict(cs472) \vee mwf(cs472, 1pm) \tag{1.11}$$

Using equations (11.9) and (11.11)

$$\begin{array}{rcl} mwf(cs472, 1pm) & \Longrightarrow & conflict(cs472) \\ T & \rightarrow & conflict(cs472) \vee mwf(cs472, 1pm) \end{array}$$

$$T \rightarrow conflict(cs472) \vee conflict(cs472)$$

Which implies

$$\mathbf{T} \rightarrow \mathbf{conflict}(\mathbf{cs472})$$

the required result.

Note:
1. This process involves search of sentences to cencel out appropriate terms.
2. This is a machenical process.

1.6 Variables and Quantifiers

We need 2 things before we can put together a general theorem proving procedure.

- Unification to handle variables etc.
- Transformation of arbitrary FOPL sentences into clause normal form(CNF)

Unification

Consider the following expressions

$$p = P(x, f(y), B)$$

$$q = P(z, f(w), B)$$

We can **unify** p and q by substituting x for z and y for w in q. The resulting substitutions are represented by a binding list (or a substitution list) of the form $\{e_i|v_i\}$ where e_i is a **term** and v_i is a **variable** (terms are constants, variables, functional expressions).
In the above example the binding list is

$$\sigma = \{x|z, y|w\}(readas``xforzandyforw'')$$

so that q$|_\sigma = p$ which means if σ is substituted in q, we get p.

Examples

(1) $p_1 = P(x, f(y), B)$ $q_1 = P(A, f(w), B)$
The substitutions are given by $\sigma = \{A|x, w|y\}$.
We can only substitute a constant for a variable but not the other way round.

(2) $p_2 = P(A, B)$ $q_2 = Q(A, B)$
p_2 and q_2 cannot be unified because P and Q are different predicates. σ is undefined for this example.

(3) $p_3 = P(A, f(x))$ $q_3 = P(A, x)$
In general they cannot be unified because f(x) contains variable x.

(4) $p_4 = P(A, f(x))$ $q_4 = P(A, y)$
Can be unified with $\sigma = \{f(x)|y\}$ We can only substitute f(x) for y but not the other way round, because f(x) could be a constant.

(5) $p_5 = P(x, y, z, f(w))$ $q_5 = P(A, y, z, f(u))$
The substitution list $\sigma = \{A|x, u|w\}$ is a Most General Unifier of p_5 and q_5. σ is more general than say $\{A|x, B|y, u|w\}$
e.g.:
$p = P[f(x, g(A, y)), g(A, y)])$ $q = P[f(x, z), z]$ the MGU (Most General Unifier) of p and q is $\{g(A, y)|z\}$.

(6) $p_6 = P(x, y, z, f(w))$ $q_6 = P(A, y, z, g(w))$
p_6 and q_6 can not be unified because f and g are different functions.

(7) $p_7 = P(A, y)$ $q_7 = P(B, y)$
p_7 and q_7 can not be unified because we can't substitute a constant for a constant.

(8) $p_8 = P(A, y)$ $q_8 = P(x, B)$
Can be unified with $\sigma = \{A|x, B|y\}$

Generally speaking, we are interested in the most general unifier (mgu) of the two expressions. So, $\sigma = \{A|x, w|y\}$ is less general than $\sigma^{'} = \{z|x, w|y\}$.

1.7 Clause Normal Form

This is the general form of resolution principle:

$$\frac{\begin{array}{l}\neg a_1 \vee \neg a_2 \cdots \neg a_{i-1} \vee \neg a_i \vee \neg a_{i+1} \cdots \neg a_n \vee b_1 \vee b_2 \cdots \vee b_k \\ \neg c_1 \vee \neg c_2 \vee \cdots \neg c_m \vee d_1 \vee d_2 \vee \cdots \vee d_{j-1} \vee d_j \vee d_{j+1} \vee \cdots \vee d_l \text{ (assume } a_{i|\sigma} = d_{j|\sigma}\text{)}\end{array}}{\begin{array}{l}(\neg a_1 \vee \neg a_2 \cdots \neg a_{i-1} \vee \neg a_{i+1} \cdots \neg a_n) \vee (b_1 \vee b_2 \vee \cdots b_k) \vee \\ (\neg c_1 \vee \neg c_2 \cdots \neg c_n) \vee (d_1 \vee d_2 \vee \cdots d_{j-1} \vee d_{j+1} \vee d_l)|_\sigma\end{array}}$$

Question:
Can we convert arbitrary FOPL sentences into a more regular form, i.e., clause normal form?

Theorem: Given a set of sentences S in FOPL, $\exists$ a set S' of sentences in *clause normal form* such that whenever S$\models q$, S'$\models q$.

Proof: We will provide an algorithm that performs this transformation with the following eample.

Example:

Original sentence: $\forall_x [B(x) \wedge H(x) \Rightarrow W(x) \vee [\exists_z M(z, x) \wedge \neg \exists_z G(z, x)]]$

1. Remove "$\Rightarrow$" using $a \Rightarrow b \Leftrightarrow \neg a \vee b$

We now have: $\forall_x [\neg(B(x) \wedge H(x)) \vee W(x) \vee [\exists_z M(z, x) \wedge \neg \exists_z G(z, x)]]$

2. Move negations down to the atomic level.

Recall the following properties:
$\neg(\neg a) \Leftrightarrow a$
$\neg(a \wedge b) \Leftrightarrow \neg a \vee \neg b$
$\neg(a \vee b) \Leftrightarrow \neg a \wedge \neg b$
$\neg(\forall_x P(x)) \Leftrightarrow \exists_x(\neg P(x))$
$\neg(\exists_x P(x)) \Leftrightarrow \forall_x(\neg P(x))$
(Infinitary versions of DeMorgan's Law)

Simplifying with these properties, we now have:
$\forall_x[\neg B(x) \vee \neg H(x) \vee [W(x) \vee \exists_z M(z,x) \wedge \forall_z \neg G(z,x)]]$

3. Standardize Variables

Standardize the variables apart so that each quantifier has a different variable associated with it. We have:
$\forall_x[\neg B(x) \vee \neg H(x) \vee W(x) \vee [\exists_z M(z,x) \wedge \forall_w \neg G(w,x)]]$

4. Eliminate '$\exists$' using Skolemization

Skolemization is explained in greater detail after this example. The idea is to replace existentially quantified variables with a unique function, a skolem function, whose variables are the universally quantified variables included in the scope of $\exists$. The following example demonstrates this.

Consider $\exists x\ housep(John, x)$

There is a house that belongs to John.
This assertion is about the existence of an object (whose identity is dependent on John) that satisfies the predicate *housep*. Suppose we imagine a function which accepts John as an argument and returns this object. Let this function be $house\_of(John)$.

Given this function, we could write:
$housep[John, house\_of(John)]$

Such functions which allow us to eliminate '$\exists$' are called Skolem functions (after the Dutch mathematician Thoralf Skolem).

In our case, $\forall x \exists z M(z,x)$ since z depends on x, we can replace z with f(x).

Using skolemization, we now have: $\forall_x[\neg B(x) \vee \neg H(x) \vee W(x) \vee [M(f(x), x) \wedge \forall_w \neg G(w, x)]]$

5. Move all universally quantified variables to the start of the expression (to the left).

We now have: $\forall_x \forall_w[\neg B(x) \vee \neg H(x) \vee W(x) \vee [M(f(x), x) \wedge \neg G(w, x)]]$

6. Drop the quantifiers (with the understanding that all variables are universally quantified).

We now have: $\neg B(x) \vee \neg H(x) \vee W(x) \vee [M(f(x), x) \wedge \neg G(w, x)]$

7. Distribute $\vee$ and $\wedge$ to write the expression as a conjunction of disjuncts.

Recall: $(a \vee (b \wedge c)) \equiv (a \vee b) \wedge (a \vee c)$

We now have: $[\neg B(x) \vee \neg H(x) \vee W(x) \vee M(f(x), x)] \wedge [\neg B(x) \vee \neg H(x) \vee W(x) \vee \neg G(w, x)]$

8. a. Drop the $\wedge$ and replace each conjunct as a separate clause.
b. Rename the variables in each clause.

We now have:
$\neg B(x) \vee \neg H(x) \vee W(x) \vee M(f(x), x)$
$\neg B(y) \vee \neg H(y) \vee W(y) \vee \neg G(z, y)$
The equations are now in *Clause Normal Form.*

1.8 More Examples of Skolemization

$\forall_y \forall_z \exists_x P(x, y, z)$ becomes $\forall_y \forall_z P(f(y, z), y, z)$

Rule: Replace each existentially quantified variable by a skolem function of those universally quantified variables that include the existential quantifier in their scope.

$[\forall w P(w)] \Rightarrow \exists z Q(z, A)$
becomes: $[\forall w P(w)] \Rightarrow Q(k, A)$, where k is a skolem constant such that a function of zero arguments f() = k.

$\forall x, y, u[\exists z[P(x, y, z) \Rightarrow R(x, y, u, z)]]$
becomes: $\forall x, y, u[P(x, y, f(x, y, u)) \Rightarrow R(x, y, u, f(x, y, u))]$

$[\forall w Q(w)] \Rightarrow \exists x \exists y \exists z [P(x,y,z) \Rightarrow \forall u R(x,y,u,z)]$
becomes: $[\forall w Q(w)] \Rightarrow [P(c_0,c_1,c_2) \Rightarrow \forall u R(c_0,c_1,u,c_2)]$
where c_0, c_1, c_2 are skolem constants

2 AUTOMATED THEOREM PROVING IN FOPL

In order to prove a theorem we negate the theorem and add it to the set of axioms. Then, by repeated application of the resolution principle, if we can derive a null clause (a contradiction), then the theorem is true. So below S entails q can be proved by adding $S \vee \neg q$ to the set of axioms and deriving a contradiction.

$S \models q \iff S \cup \neg q$ resolves to null clause.

Example:
If a course is interesting, some students are happy.
if a course has a final, no student is happy.
Prove: If a course has a final, then it is not interesting.

Putting this in FOPL we get:

1. $\forall c Interesting(c) \Rightarrow \exists s[Student(s,c) \wedge Happy(s)]$
2. $\forall s \forall_c [Final(c) \wedge Student(s,c) \Rightarrow \neg Happy(s)]$

Theorem to prove : $\forall c Final(c) \Rightarrow \neg Interesting(c)$
Negation of theorem :
3. $\neg[\forall c Final(c) \Rightarrow \neg Interesting(c)]$

By inspection we can translate the above into clause normal form :
a. $\neg Interesting(c) \vee Student(skf(c), c)$
b. $\neg Interesting(x) \vee happy(skf(x))$
c. $\neg Final(z) \vee \neg Student(s,z) \vee \neg Happy(s)$
d. $Final(sk\phi)$
e. $Interesting(sk\phi)$

a. $\neg Interesting(c) \vee Student(skf(c), c)$
e. $Interesting(sk\phi)$ $\sigma = \{sk\phi | c\}$

f. $[Student(skf(sk\phi), sk\phi)]$
c. $\neg Final(z) \vee \neg Student(s, z) \vee \neg Happy(s)$ $\sigma = \{skf(sk\phi)|s, sk\phi|z\}$

g. $[\neg Final(sk\phi) \vee \neg Happy(skf(sk\phi))]$
d. $Final(sk\phi)$ $\sigma = \{\}$

h. $\neg Happy(skf(sk\phi))$
b. $\neg Interesting(x) \vee Happy(skf(x))$ $\sigma = \{sk\phi|x\}$

i. $\neg Interesting(sk\phi)$
e. $Interesting(sk\phi)$

[Null clause]

Therefore we have a contradiction and so we have a proof that if a course has a final, then it is not interesting.

3 SEARCH CONTROL IN THEORM PROVING

We know that repeated applications of the resolution inference rule will find a proof if one exits, but we have no guarantee of the efficiency of this process. In this section we look at several stategies that have been used to guide the search toward a proof.

3.1 Unit Preference

This strategy prefers to do resolutions where one of the sentences is a single literal (also known as a **unit clause**). The idea behind the strategy is that we are trying to produce a very short sentence, True $\Rightarrow$ False, and therefore if possible pick one of the clauses that has a single literal at each reaolution step.

Example: Consider the clauses p;
$\neg p \vee \neg q \vee r$; and $p \vee r$. In this case, unit preference results in selection of the first clause over the third as a candidate for resolution against the second clause.

3.2 Set of Support

Definition: A clause $c_j \in$ set of support if $c_j \in$ Negated Theorem(set of clauses that correspond to the negated theorm) or at least one parent of $c_j \in$ set of support.

At each step, one of the parent clauses is chosen from the set of support.

Example:
Axioms: I(A), D(A), $\neg R(x) \vee L(x)$, $\neg D(y) \vee \neg L(y)$
Negated theorem: $\neg I(z) \vee R(z)$

While running inference, at least one of the clauses is in the set of support.
$\neg I(z) \vee R(z)$
I(A) $\sigma = \{A|z\}$

$R(A)$
$\neg R(x) \vee L(x)$ $\sigma = \{A|x\}$

$L(A)$
$\neg D(y) \vee \neg L(y)$ $\sigma = \{A|y\}$

$\neg D(A)$
D(A)

[Null clause]

Theorem: Set of support search control strategy is complete (it is guaranteed to derive a null clause using the axioms and negated theorem, whenever the theorem is true) when used with resolution principle.

3.3 Other Simplification Strategies

Clauses that have certain properties can be eliminated even before they are even considered as candidates for resolution.

1. Always eliminate tautologies.
e.g. Clause of the form $D(x) \vee \neg D(x)$ can be disregarded during the proof

2. Eliminate clauses that contain "pure" literals (those literals whose negation does not appear in any other clause).
e.g.
a. $I(z)$
b. $D(z)$
c. $\neg L(x) \vee P(x)$
d. $\neg I(A)$
e. $\neg P(A)$
Clauses b and c are eliminated since $\neg D(z)$ and $L(x)$ do not appear in any other clause.
This will not affect the soundness of the proof procedure.

3. Eliminate clauses that are subsumed by other clauses.

Definition: A clause ϕ *subsumes* a clause ψ iff $\exists$ a substitution σ such that $\phi|_\sigma \subseteq \psi$

Examples:
1.$P(x)$ subsumes $\{P(x), D(y)\}$
i.e. $\{P(x)\}|_{\sigma=\{\}} \subseteq \{P(x), D(y)\}$
2. $P(x) \vee Q(y)$ subsumes $P(f(A)) \vee Q(A) \vee R(z)$ where $\sigma = \{f(A)|x, A|y\}$

Theorem: A set of clauses S' that is obtained by eliminating every clause c' that is subsumed by some other clause c in a set of clauses S is unsatisfiable iff S' is unsatisfiable.
Proof:
Let S be a set of clauses such that $S \models q$
Suppose $S = \{c_1, c_2, c_3, ..., c_n, c, c'\}$
Let $S' = \{c_1, c_2, ..., c_n, c\} = S - \{c'\}$
Let $S'' = \{c_1, c_2, ..., c_n\} = S - \{c, c'\} = S' - \{c\}$
Let c subsumes c'

WLOG, let $c = l_1$, $c' = l_1 \vee l_2$
(for the time being, assume that l_1 and l_2 are ground literals.)
Let M_s = the set of models in which S holds.
Let M_s' = the set of models in which S' holds.

Let $M_s'' =$ the set of models in which S'' holds.

So, in every model $m \in M_s$
$c_1 \wedge c_2 \wedge c_3 ... \wedge c_n \wedge c \wedge c'$ must hold.
or, S'' must hold and c and c' must hold.
Let $M_c =$ set of models in which c holds.
Let $M_c' =$ set of models in which c' holds.

Since c subsumes c', $c_1 = l_1$ and $c' = l_1 \vee l_2$
or $M_c \cap M_c' = M_c$

$M_s =$ the set of models in which S holds.
$M_s = M_s'' \cap M_c \cap M_c' = M_s'' \cap M_c = M_s'$

Thus S' holds iff S holds. That means if $S \models q$, $S' \models q$ and vice versa. $\Box$

Thus we can eliminate clauses subsumes by other clauses in any refutation-complete search strategy.

In set of support, we need to ensure that if the clause eliminated was a member of the set of support, but the other was not, the latter needs to be added to the set of support to guarantee completeness.

Observation: FOPL is semi-decidable
If a theorem locically follows from a set of axioms, then a proof can be found in a finite time. But if a theorem does not follow from the axioms, the search may not terminate. Contrast this with propositional logic which is decidable for any finite propositional language.

4 GREEN'S TRICK FOR ANSWER EXTRACTION

In many applications (e.g., deductive databases), we are interested not simply in proving some assertion about some entities, but in finding instances of such entities that make the assertion true. For instance, consider the assertion $\exists x At(Daisy, x)$ which we can prove from the following axioms: $\forall_x At(Bumstead, x) \Rightarrow$
$At(Daisy, x)$
$At(Bumstead, couch)$

What if we wanted to also know Daisy's current whereabouts. We can think of this in terms of answering the query: Query : $At(Daisy, z)?$.

This can be accomplished with a simple trick (called Green's trick) as follows. We prove the theorem $\exists z At(Daisy, z)$ as usual using resolution by refutation. In parallel, we start with the query we want answered $At(Daisy, z)$ and apply every substitution used in proving the theorem in exactly the same order, into the query expression. The readers are encouraged to verify that this transforms the query expression into $At(Daisy, Couch)$, thereby answering the query. Green's trick finds use in deductive databases and question answering systems.