CITS3005 Knowledge Representation

Lecture 2: Foundations of Logic

Tim French

The University of Western Australia

2023

Overview

In this lecture we will look at logic in its broadest sense and consider how a logic can be formalised and implemented in a machine.

Logic, as the methodology of reason has been studied for thousands of year, but logic as a formal process was studied intensely in the early twentieth century, as a response to Hilbert's questions:

Consistency of Arithmetic

Show that the axioms of arithmetic are consistent.

Decision Problem

Show that there is an effective process for determining whether a statement is provable in first order logic.

In this unit we will see several different logics, so here we aim to set up a common framework for discussing logic.

Elements of Logic

In the most general sense, a logic requires:

- *Propositions*: symbols or statements that can be considered true or false within some frame of reference, like "it rained today", or "Kate is tired".
- *Consequence*: a set of rules that determine that one proposition follows from one or more other propositions. From example, from "it rained today" derive "the grass is wet". There is no requirement for the rule to be factually valid.

Logic can be expressed in general terms: *A car with a flat battery will not start. The car started so the battery is not flat.*

Logic can be expressed in formal terms:

$$\forall x \in \mathbb{R},\ x^2 \geq 0 \implies \pi^2 \geq 0.$$

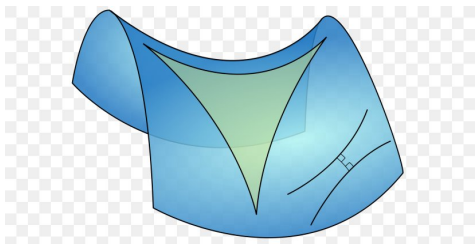
Elements of Formal Logic

In formal logic propositions are given a grammatical structure, and consequence is defined with respect to that structure.
This divides the logic into:

1. *Syntax*: how the propositions are defined from atoms and operators.
2. *Semantics*: the meaning of the propositions.

The define semantics the meaning must be grounded in some interpretation. Formal logic uses mathematical structures as the basis for the interpretation.

Note, there is an alternative *proof-theoretic semantics* that characterise the meaning of propositions through rules of inference, rather than through interpretations.
A famous example is *non-Euclidean geometry* for which the existence of an interpretation was a long standing open question.



Example: Propositional Logic

The language of propositional logic consists of

- a set P of propositional atoms: `rain_today`, `Kate_is_tired`, `x`, ...
- the constants $\top$ (true) and $\bot$ (false)
- the operators: $\neg$ (not), $\wedge$ (and), $\rightarrow$ (implies), $\leftrightarrow$ (if and only if).

The set of propositions is specified using Backus Naur form (BNF):

$$\alpha ::= \top \mid \bot \mid x \mid \neg\alpha \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \mid \alpha \rightarrow \alpha \mid \alpha \leftrightarrow \alpha$$

where $x \in P$.
Any terms defined as α can be substituted for α in the grammar, so this will generate an infinite set of propositions:

$$x \wedge y,\ \neg z \rightarrow (x \wedge y), \neg(x \vee z) \rightarrow (x \wedge y), \ldots$$

Note that BNF imposes a tree structure on a formula, so brackets aren't required in the formal syntax, but we use them to write the formulas as strings.

Example Continued: Propositional Logic

An interpretation of propositional logic is simply the set $\mathcal{I} \subseteq P$ or atomic propositions that are true.
The *semantics* of propositional logic determine whether a proposition, α is true, given an interpretation, $\mathcal{I}$. We say $\mathcal{I} \models \alpha$, and define this relation recursively:

$$
\begin{aligned}
\mathcal{I} &\models \top \\
\mathcal{I} &\not\models \bot \\
\mathcal{I} \models \neg\alpha &\quad\Leftrightarrow\quad \mathcal{I} \not\models \alpha \\
\mathcal{I} \models \alpha \wedge \beta &\quad\Leftrightarrow\quad \mathcal{I} \models \alpha \text{ and } \mathcal{I} \models \beta \\
\mathcal{I} \models \alpha \vee \beta &\quad\Leftrightarrow\quad \mathcal{I} \models \alpha \text{ or } \mathcal{I} \models \beta \\
\mathcal{I} \models \alpha \rightarrow \beta &\quad\Leftrightarrow\quad \mathcal{I} \models \beta \text{ or } \mathcal{I} \not\models \alpha \\
\mathcal{I} \models \alpha \leftrightarrow \beta &\quad\Leftrightarrow\quad \mathcal{I} \models \alpha \rightarrow \beta \text{ and } \mathcal{I} \models \beta \rightarrow \alpha
\end{aligned}
$$

There is a degree of redundancy in these operators as $\alpha \vee \beta$ is equivalent to $\neg(\neg\alpha \wedge \neg\beta)$, $\alpha \implies \beta$ is equivalent to $\neg\alpha \vee \beta$ etc so $\wedge$ and $\neg$ are normally sufficient to describe the full semantics.

Validity and Satisfiability

We say a formula is *valid* if it is true in every possible interpretation, and we say a formula is *satisfiable* if it is true in some possible interpretation.

The fundamental definition of a logic is the set of all valid formulas.

However, determining whether a formula is valid or not is generally computationally intractable, so this is not necessarily the best approach to reasoning.

x	y	z	$\neg(x \wedge y) \rightarrow z$
$\top$	$\top$	$\top$	$\top$
$\top$	$\top$	$\bot$	$\top$
$\top$	$\bot$	$\top$	$\top$
$\top$	$\bot$	$\bot$	$\bot$
$\bot$	$\top$	$\top$	$\top$
$\bot$	$\top$	$\bot$	$\bot$
$\bot$	$\bot$	$\top$	$\top$
$\bot$	$\bot$	$\bot$	$\bot$

Proof and Derivation

While the semantic interpretations allow us to understand a formulas in terms of a mathematical description of an interpretation, it is often the case that it is intractable to build a full interpretation.

A "proof" progresses from what we *know* to what we can *infer*: in that sense we maintain a *knowledge base* and add new knowledge from either observation or by applying deduction.

Deductive systems come in several forms:

- Hilbert-style axiom systems
- Sequent calculus
- Natural Deduction
- Tableaux

Deductive Systems

Deductive systems progress from what we know to what we can infer:

$$(\wedge\text{-intro})\frac{\alpha, \quad \beta}{\alpha \wedge \beta} \qquad (\text{modus ponens})\frac{\alpha, \quad \alpha \to \beta}{\beta}$$

Hilbert style axiom systems give fundamental axioms (e.g. $x \vee \neg x$) and provide rules to add to the set of axioms (like substitution: *from* $x \vee \neg x$ *infer* $\mathit{raining} \vee \neg \mathit{raining}$).

A proof is a sequence of deductive steps from a set of premises, Θ, (or *antecedents*) to a conclusion, α (or *consequent*) and we write $\Theta \vdash \alpha$ to say that α *is derivable from* Θ

$$\begin{aligned}
\alpha \wedge \beta & \qquad \text{Premise} \\
\vdash \alpha \wedge \beta \to \alpha & \qquad \wedge\text{-elimination} \\
\alpha \wedge \beta \vdash \alpha & \qquad \text{modus ponens} \\
\vdash \alpha \to \alpha \vee \beta & \qquad \vee\text{-introduction} \\
\alpha \wedge \beta \vdash \alpha \vee \beta & \qquad \text{modus ponens} \\
\vdash (\alpha \wedge \beta) \to (\alpha \vee \beta) & \qquad \to\text{-introduction}
\end{aligned}$$

Natural Deduction

Natural deduction attempts to codify intuitive rules that can present an argument in a natural way way.

Modus Ponens (MP) p ⇒ q p _____ ⊢ q	Modus Tollens (MT) p ⇒ q ~q _____ ⊢ ~p
Hypothetical Syllogism (HS) p ⇒ q q ⇒ r _____ ⊢ p ⇒ r	Disjunctive Syllogism (DS) p ∨ q ~p _____ ⊢ q
Constructive Dilemma (CD) (p ⇒ q) ∧(r ⇒ s) p ∨ r _____ ⊢ q ∨ s	Destructive Dilemma (DD) (p ⇒ q) ∧(r ⇒ s) ~q ∨ ~s _____ ⊢ ~p ∨ ~r
Simplification (Simp.) p∧q _____ ⊢ p	Conjunction (Conj.) p q _____ ⊢ p∧q
Addition (Add.) p _____ ⊢ p ∨ q	

De Morgan's Theorem (DM)	~(p∧q) ⇔ (~p ∨ ~q) ~(p ∨ q) ⇔ (~p∧~q)
Commutation (Com.)	(p∨q) ⇔ (q∨p) p∧q ⇔ q∧p
Association (Assoc.)	[(p∨q)∨r] ⇔ [p∨(q∨r)] [p∧(q∧r)] ⇔ [(p∧q)∧r]
Distribution (Dist.)	[p∧(q∨r)] ⇔ [(p∧q) ∨ (p∧r)] [p ∨ (q∧r)] ⇔ [(p ∨ q)∧(p ∨ r)]
Double Negation (DN)	p ⇔ ~~p
Material Implication (M. Imp.)	(p ⇒ q) ⇔ (~p ∨ q)
Transposition (Trans.)	(p ⇒ q) ⇔ (~q ⇒ ~p)
Material Equivalence (M. Equiv.)	(p ⇔ q) ⇔ [(p ⇒ q)∧(q ⇒ p)] (p ⇔ q) ⇔ [(p∧q) ∨ (~p∧~q)]
Law of Exportation	[(p∧q) ⇒ r] ⇔ [(p ⇒ (q ⇒ r)]

For computational reasoning it is often more efficient to use less explainable rules with more efficient implementations, like resolution:

$$\frac{\Theta, \alpha \vdash \beta \qquad \Psi, \neg\alpha \vdash \beta}{\Theta, \Psi \vdash \beta}$$

or conversion to conjunctive normal form.

Consistency, Soundness and Completeness

Once we have a deductive system we can ask the following questions:

- Is it consistent? That is, are all the axioms true in every interpretation?
- Is it sound? That is, do the rules produce conclusions that are true in every interpretation?
- Is it complete? That is, can it prove *every* valid proposition?

Other logical questions are:

- Satisfiability checking: can we check whether a given formula has a model, and if so, how complex is the process.
- Model checking: given an interpretation, check whether a proposition holds in that interpretation.
- Expressivity: given two logics can one be translated to the other?
- Synthesis: given a proposition, can we construct an interpretation where that proposition is true.

Types of Logic

There are many different types of logic, including;

- propositional logic
- first order logic (and second order logic,...)
- description logic
- modal logics (logics or time, space, knowledge, obligation, necessity,...)
- computational logics (logics of program correctness
- logics of uncertainty (fuzzy logic, probabilistic logic, para-consistent logic)

Logic programming is a fragment of first order logic with a computational implementation.

First Order Logic Overview

We will now consider the formal aspects of *first order logic*, including the syntax, the semantics, and reasoning techniques.

First order logic goes beyond propositional logic, where every proposition can only include a finite number of Boolean concepts, to consider unbounded sets of entities.

Functions describe correspondences between these entities and predicates define relations over these entities. First order logic uses quantifiers to define propositions that are dependent on the entire domain, rather than propositions that are isolated facts.

First order logic is the logic of ontology and artificial intelligence. It is just expressive enough to describe the state of the world, what *is*.

Elements of First Order Logic

The elements of first order logic consists of :

- *Entities*: elements of the domain, about which we have facts. For example, *tim, jane, cits3005, uwa*. Some may have a fixed label (a *constant*), like *tim_french*, but we can also reason about sets of entities without needing each one to be labelled (e.g. all the students at UWA).
- *Functions*: these map tuples of entities to other entities. For example $grade(jane, cits3005) = credit$ is a function mapping students and units to grades.
- *Predicates*: these provide the truth function for the logic, and assign tuples of entities a truth assignment, with meaning. For example, $enrolled(jane, cits3005)$, is true if *jane* is *enrolled* in *cits3005* and false otherwise.
- *Operators*: these are just the operators from propositional logic, $\wedge$, $\vee$, $\neg$, etc
- *Quantifiers*: $\exists$ (*there exists*) and $\forall$ (*for all*), that range over all entities.

Language of First Order Logic

The *language* of a first order logic is given by:

- A set C of *constant symbols*, eg $tim \in C$;
- A set F of *function symbols*, where each $f \in F$ has arity $\#f$, eg $grade \in F$ and $\#grade = 2$;
- A set P of *predicate symbols*, where each $p \in P$ has arity $\#p$, eg $enrolled \in P$ and $\#enrolled = 2$;
- A set var of *variable symbols*, where we write variables starting with a capital, eg $X \in \mathrm{var}$.

Different domains will use different languages. For example:

- Family Tree: $C = \{john, jane, cris...\}$, $F = \{mother, father\}$, $P = \{ancestor, male, related\}$.
- Arithmetic: $C = \{0, 1\}$, $F = \{successor, +\}$, $P = \{=, \leq\}$

Syntax of First Order Logic

The syntax of first order logic has two parts: the *terms* correspond to the entities; and the *propositions* correspond to truth values.

The terms are given by the following BNF:

$$\tau \ ::= \ c \mid X \mid f(\tau_1, \ldots, \tau_{\#f})$$

where $c \in C$, $X \in \mathrm{var}$ and $f \in F$.

The propositions are then given by the BNF:

$$\alpha \ ::= \ \top \mid \neg\alpha \mid \alpha \wedge \alpha \mid p(\tau_1, \ldots, \tau_{\#p}) \mid \forall X \alpha$$

where $X \in \mathrm{var}$, $p \in P$, and τ_i is a term.

We use the abbreviations $\bot = \neg\top$, $\alpha \vee \beta = \neg(\neg\alpha \wedge \neg\beta)$, $\alpha \to \beta = \neg\alpha \vee \beta$, and $\exists X \alpha = \neg\forall X \neg\alpha$.

If a variable X in α does not appear in the scope of a quantifier, $\forall X$, then it is a *free variable*, and a formula with no free variables is a *sentence*.

Exercise: Logics of Arithmetic

Express some simple arithmetic properties using arithmetic logic:

- Every even number is followed by an odd number.
- For every number there is always a prime larger than it
- Every number is the square of some different number.

Interpretations of First Order Logic

The semantics of first order logic over a language (C, F, P, var), are given with respect to an interpretation, $\mathcal{I}$, consisting of:

- a *domain*, $D^{\mathcal{I}}$, which is the set of entities in the system.
- for every constant symbol, $c \in C$, an element of the domain $c^{\mathcal{I}} \in D^{\mathcal{I}}$.
- for every function symbol, $f \in F$, a function $f^{\mathcal{I}} : (D^{\mathcal{I}})^{\#f} \longrightarrow D^{\mathcal{I}}$
- for every predicate symbol, $p \in P$. a function $p^{\mathcal{I}} : (D^{\mathcal{I}})^{\#p} \longrightarrow \{\top, \bot\}$

However, the interpretation doesn't tell us anything about the free variables, as they are not yet bound to the element of a domain.
We define an assignment to be a map from variables to domain elements: $\theta : \mathrm{var} \longrightarrow D^{\mathcal{I}}$, and extend θ to act on all terms by: $\theta(c) = c^{\mathcal{I}}$ for all $c \in C$ and $\theta(f(t_1, \ldots, t_{\#f})) = f^{\mathcal{I}}(\theta(t_1), \ldots, \theta(t_{\#f}))$.

Semantics of First Order Logic

We can now give the semantics for first order logic.
We say $\mathcal{I}$ *satisfies* α *given the assignment* θ (written $\mathcal{I}^\theta \models \alpha$), and define this relation recursively:

$$\mathcal{I}^\theta \models \top$$

$$\mathcal{I}^\theta \models p(\tau_1, \ldots, \tau_{\#p}) \iff p^{\mathcal{I}}(\theta(\tau_1), \ldots \theta(\tau_{\#p})) = \top$$

$$\mathcal{I}^\theta \models \neg\alpha \iff \mathcal{I}^\theta \not\models \alpha$$

$$\mathcal{I}^\theta \models \alpha \wedge \beta \iff \mathcal{I}^\theta \models \alpha \text{ and } \mathcal{I}^\theta \models \beta$$

$$\mathcal{I}^\theta \models \forall X \alpha$$
$$\iff \forall \psi : \text{var} \to D^{\mathcal{I}} \text{ s.t. } \theta(Y) = \psi(Y) \text{ if } X \neq Y,\ \mathcal{I}^\psi \models \alpha.$$

Example: University

For example, in a University:

- the domain might be students, lecturers and units.
- the constants could be *tim*, *jane*, *haolin*, *cits3005*, *cits2211*.
- a function might be *uc* that maps a unit to the lecturer that coordinates it.
- predicates might be *studies*, that relates students to the units that study, or *prereq* that relates a unit to its prerequisite, or $=$ for when two domain elements are equal.

We can then considered sentences like:

$$\forall \texttt{Units}\ (\mathit{uc}(\texttt{Unit}) = \mathit{tim} \rightarrow \exists \texttt{Student}\ \mathit{studies}(\texttt{Student}, \texttt{Unit}))$$

For every unit that is coordinated by Tim, there is a student that studies that unit.

Example: Natural Numbers

First order logic can also express properties of arithmetic:

- The domain is the natural numbers $0, 1, 2, 3, 4, \ldots$
- the only constant is 0 (zero).
- There is a function *succ* that maps a number to its successor, so $succ(1) = 2$, and there are also functions $+$, $-$, $/$, $*$ etc.
- There are predicates $=$, $<$, , $\leq$.

We can define sentences like:

$$\forall X \forall Y (X < Y \rightarrow succ(X) < succ(Y))$$

or

$$\forall X \exists Y (X < Y \wedge \forall Z (Z < Y \rightarrow (\exists W (W * Z = Y)) \rightarrow Z = succ(0))).$$

Validity and Satisfiability

We say a formula is *valid* if it is true in every possible interpretation, and we say a formula is *satisfiable* if it is true in some possible interpretation.

In first order logic, determining whether a formula is valid or not is *not decidable*. This means that there is no computational process that can definitively answer the question.

Validity in first order logic is *semi-decidable*, which means there is a process, such that, given long enough, will compute every possible validity of first order logic.
However to know something is *not* valid, would require us to wait for the generation process to end, and it is an infinite process.
One such process for generating all validities is to apply natural deduction.

A Natural Deduction System for First Order Logic

true	$\frac{}{\top}$	*false*	$\frac{\alpha \wedge \neg\alpha}{\bot}$
$\neg$-*intro*	$\frac{\alpha \to \bot}{\neg\alpha}$	$\neg$-*elim*	$\frac{\neg\neg\alpha}{\alpha}$
$\wedge$-*intro*	$\frac{\alpha \quad \beta}{\alpha \wedge \beta}$	$\wedge$-*elim*	$\frac{\alpha \wedge \beta}{\alpha \quad \beta}$
$\to$ -*intro*	$\frac{\begin{array}{c}\alpha \\ \vdots \\ \beta\end{array}}{\alpha \to \beta}$	$\to$ -*elimination*	$\frac{\alpha \quad \alpha \to \beta}{\beta}$
$\exists$-*intro**	$\frac{\alpha(t)}{\exists X \alpha(X)}$	$\exists$-*elim*†	$\frac{\beta(t) \to \alpha}{\exists X \beta(X) \to \alpha}$
$\forall$-*intro*†	$\frac{\alpha \to \beta(t)}{\alpha \to \forall X \beta(X)}$	$\forall$-*elim**	$\frac{\forall X \alpha(X)}{\alpha(t)}$

*: here t cannot occur with think the scope of a quantifier over any sub-term of t (t must be *free* in α).

†: here, α cannot contain the term t or any of its sub-terms, and t cannot be contained within an undischanged assumption.

Example Deduction

Suppose we wanted to apply natural deduction to show:

$$\text{From } \forall X(\alpha(X) \to \beta(X) \text{ we can infer } \exists X\alpha(X) \to \exists(X)\beta(X)$$

$\forall X(\alpha(X) \to \beta(X)$	assumption 1	1.
$\alpha(t) \to \beta(t)$	$\forall -$ elim, 1	2.
$\alpha(t)$	Assumption 2	3.
$\beta(t)$	$\to$ -elim, (2, 3)	4.
$\exists X\beta(X)$	$\exists$-intro, (4)	5.
$\alpha(t) \to \exists X\beta(X)$	$\to$ -intro, (3, 5)	6.
$\exists X\alpha(X) \to \exists X\beta(X)$	$\exists$-elim, (6)	$\square$

Automated Reasoning

Formal reasoning is difficult and error prone, so it is most useful to us if it can be automated and verified. From an implementation point of view, it is most convenient to work with a few rules and a restricted syntax, so reasoning can be mechanically applied in a uniform fashion.

The reasoning problems we are most interested in are

- *satisfiability* - is there a satisfying interpretation for a set of formulas
- *validity* - does one formula always follow from a set of formulas.

These problems are duals of one another (i.e. α follows from Θ if and only if $\Theta \cup \{\neg\alpha\}$ is *not* satisfiable.).

Therefore we can transform formulas in without preserving their exact meaning, so long as we preserve the existence of a satisfying model.

We will give such a transformation to *clausal normal form*.

Conjunctive Normal Form

A formula in conjunctive normal form is specified by the syntax:

$$\begin{array}{rcl} \tau & ::= & c \mid X \mid f(\tau_1, \ldots, \tau_{\#f}) \\ \texttt{lit} & ::= & P(\tau_1, \ldots, \tau_{\#P} \mid \neg P(\tau_1, \ldots, \tau_{\#P} \mid \top \mid \bot \\ \texttt{disj} & ::= & \texttt{lit} \mid \texttt{disj} \vee \texttt{lit} \\ \texttt{conj} & ::= & \texttt{disj} \mid \texttt{conj} \wedge \texttt{disj} \end{array}$$

To convert a formula into conjunctive normal form we need to:

1. push all the negations ($\neg$) down to the literals.
2. remove all of the existential quantifiers.
3. remove all of the universal quantifiers.
4. move all of the conjunctions to the outside of the formula.
5. remove all of the true/false constants.

Moving negations to literals.

To move the negations to literals, we can apply the following transformations recursively:

$$
\begin{aligned}
\neg(\alpha \wedge \beta) &\iff \neg\alpha \vee \neg\beta \\
\neg(\alpha \vee \beta) &\iff \neg\alpha \wedge \neg\beta \\
\neg\exists X \alpha &\iff \forall X \neg\alpha \\
\neg\forall X \alpha &\iff \exists X \neg\alpha \\
\neg\neg\alpha &\iff \alpha \\
\neg\top &\iff \bot \\
\neg\bot &\iff \top
\end{aligned}
$$

Since on the right side of each equality, the negated formula is less complex than the negated formula on the right, we can apply these transformations until negations are only applied to predicates.

Quantifier Elimination 1: Skolemisation

Suppose we are given a formula $\beta(\exists X \alpha(t_1, ..., t_n, X))$. That is the formula β contains a subformula $\exists X \alpha(\ldots)$, and α contains the *unquantified* terms $t_1, ...t_n$ as well as the variable X.

$\exists X \alpha(t1, ..., t_n, X)$ is true if given $t_1, ..., t_n$ we can find an X that makes α true. That is, X is dependent on $t_1, ..., t_n$, so we can introduce a new function to represent this dependency:

$$\beta(\exists X \alpha(t_1, ..., t_n, X)) \cong \beta(\alpha(t_1, ...t_n, f_X(t_1, ..., t_n)))$$

This process is called *Skolemisation* and can be applied to every existentially quantified sub-formula.
Note, it is not an axiom, since it assumes the function f_X exists, but given any interpretation, we can always add an extra such function.

Quantifier Elimination 2: Generalisation

Suppose we are given a formula $\beta(\forall X \alpha(t_1, ..., t_n, X))$.
Since unquantified variable X has no constraints placed upon it, its interpretation agrees with a universally quantified variable (see $\forall$-intro).
Therefore, we can just remove the universal quantifiers of a formula in clausal normal form.

Example: suppose our domain is the natural numbers, and we have the formula:

$$\exists X \forall Y \exists Z (X - Z = Z - Y)$$

Letting *eq* be a predicate, and *sub* be a function we have:

$$\begin{aligned}
&\exists X \forall Y \exists Z (eq(sub(X, Z), sub(Z, Y)) \\
&\implies \exists X \forall Y (eq(sub(X, f(X, Y)), sub(f(X, Y), Y)) \\
&\implies \forall Y (eq(sub(x, f(x, Y)), sub(f(x, Y), Y)) \\
&\implies (eq(sub(x, f(x, Y)), sub(f(x, Y), Y))
\end{aligned}$$

That is: $x - f(x, Y) = f(x, Y) - Y$. This predicate is not true for the natural numbers, but taking $f(x, y) = \frac{x+y}{2}$ would make it true for the rational numbers.

Conjunctive Normal Form

The above transformations will restrict formulas to the syntax:

$$\begin{array}{rcl} \tau & ::= & c \mid X \mid f(\tau_1, \ldots, \tau_{\#f}) \\ \texttt{lit} & ::= & P(\tau_1, \ldots, \tau_{\#P} \mid \neg P(\tau_1, \ldots, \tau_{\#P} \mid \top \mid \bot \\ \texttt{prop} & ::= & \texttt{lit} \mid \texttt{prop} \vee \texttt{prop} \mid \texttt{prop} \wedge \texttt{prop} \end{array}$$

So we still need to transform the formula so that disjunctions only apply to literals and other disjunctions. This can be done as:

$$\begin{array}{rcl} \alpha \vee (\beta \wedge \gamma) & \iff & (\alpha \vee \beta) \wedge (\alpha \vee \gamma) \\ \top \vee \beta & \iff & \top \\ \bot \vee \beta & \iff & \beta \\ \top \wedge \beta & \iff & \beta \\ \bot \wedge \beta & \iff & \bot \end{array}$$

Now we have a formula in *clausal normal form*, and this formula is satisfiable if and only if the original formula is satisfiable.

Knowledge Bases

We can now reduce any set of first order formulas to one large formula in clausal normal form.

$$\bigwedge \begin{array}{l} (\neg)p_1(t_{11},\ldots,t_{1\#p_1}) \vee \ldots \vee (\neg)p_n(t_{n1},\ldots,t_{n\#p_n}) \\ (\neg)q_1(u_{11},\ldots,u_{1\#q_1}) \vee \ldots \vee (\neg)q_n(u_{n1},\ldots,u_{n\#q_n}) \\ \vdots \\ (\neg)r_1(v_{11},\ldots,v_{1\#r_1}) \vee \ldots \vee (\neg)r_n(v_{n1},\ldots,v_{n\#r_n}) \end{array}$$

Each clause

$$\neg p(t_{11},\ldots,t_{1\#p_1}) \vee \neg p(t_{n1},\ldots,t_{n\#p_n}) \vee q(u_{11},\ldots,u_{1\#q_1}) \vee q(u_{n1},\ldots,u_{n\#q_n})$$

can be rewritten as

$$(p(t_{11},\ldots,t_{1\#p_1}) \wedge p(t_{n1},\ldots,t_{n\#p_n})) \rightarrow (q(u_{11},\ldots,u_{1\#q_1}) \vee q(u_{n1},\ldots,u_{n\#q_n}))$$

which is often written, (using ASCII syntax) as, for example:
`q1(u1,u2), q2(f1(u3),u4) :- p1(t1,t2,f(t2)), p2(t1).`
A *knowledge base* typically describes the set of facts and rules known about a system, where facts are clauses with no negative literals:
`blue(sky).`
And rules are clauses with at least one negative literal:
`day(time) :- blue(sky).`

Resolution

Resolution is a reasoning process that uses a single rule, that is applied to two clauses:

$$\frac{\begin{array}{c} a, b \leftarrow c, d \\ c, e \leftarrow f, g \end{array}}{a, b, e \leftarrow f, g, d}$$

In general, given two clauses:

$$p_1(t_{11}, \ldots), p_n(t_{n1}, \ldots), \ldots \leftarrow q_1(u_{11}, \ldots), \ldots, q_m(u_{m1}, \ldots$$

and

$$q_1(u_{11}, \ldots), r_1(v_{11}, \ldots), \ldots \leftarrow s_1(w_{11}, \ldots), \ldots, s_n(w_{m'1}, \ldots)$$

with a common element, we can combine them into a single clause:

$$\begin{array}{c} p_1(t_{11}, \ldots), p_n(t_{n1}, \ldots), \ldots, r_1(v_{11}, \ldots), \ldots \\ \leftarrow \\ q_2(u_{21}, \ldots), \ldots, q_m(u_{m1}, \ldots), s_1(w_{11}, \ldots), \ldots, s_n(w_{m'1}, \ldots) \end{array}$$

However, we have universal variables in our knowledge base, so it's not clear when we can apply resolution.

Unification

Suppose we are given two predicates, with the same predicate symbol, but different terms. We would like to know under what substitutions the two predicates will be equal.
A substitution $\theta : \mathrm{var} \to \mathrm{terms}$ assigns a term to each variable.
Given a predicate $p(t_1, \ldots, t_{\#p})$ and a substitution θ, $p(t_1, \ldots, t_{\#p})\backslash\theta$ is the result of substituting every variable X that appears in any term in $p(t_1, \ldots, t_{\#p})$, with $\theta(X)$.

For example,

$$\mathrm{drive}(\mathrm{father}(X), Y)\backslash[X : \mathrm{sam}, Y : \mathrm{morris}]$$

becomes

$$\mathrm{drive}(\mathrm{father}(\mathrm{sam}), \mathrm{morris})$$

To *unify* two predicate statements, p_1, p_2 with the same predicate symbol is to find two substitutions, θ_1, θ_2 so that $p_1\backslash\theta_1$ and $p_2\backslash\theta_2$ are identical.

Example

For example, given $\mathrm{drive}(\mathrm{father}(X), Y)$ and $\mathrm{drive}(X, \mathrm{car}(X))$ then:

$$\mathrm{drive}(\mathrm{father}(X), Y)\backslash[X : \mathrm{sam}, Y : \mathrm{car}(\mathrm{father}(\mathrm{sam}))]$$
$$\mathrm{drive}(X, \mathrm{car}(X))\backslash[X : \mathrm{father}(\mathrm{sam})]$$
$$\mathrm{drive}(\mathrm{father}(\mathrm{sam}), \mathrm{car}(\mathrm{father}(\mathrm{sam})))$$

are all equal.

The most general unifier is the pair of substitutions that makes the predicates equal, with the least commitment.

Exercises

Ex 1. Show that if a unifier exists, a most general unifier exists.

Ex 2. Give an algorithm for finding the most general unifier.

Ex 3. Can you find a most general unifier for $p(X, f(X))$ and $p(f(Y), Y)$.

Examples

Given the statements:

- Cris is a student in CITS3005.
- All students who study hard will pass if a unit is easy.
- CITS3005 is easy.
- Cris studies hard.

Formally show Cris passes CITS3005.

Logic Programming

A logic program is essentially just a set of of clauses, and an execution of a logic program is a process of unification and resolution.

Logic programming languages add special predicates for reading and writing input, and other functions, and provide extra syntax, like cuts, for controlling execution.

Logic programs also typically restrict the clauses to have a single positive literal, to constrain the possible executions of a program.

We will consider the finer details of logic programming in the next lecture.