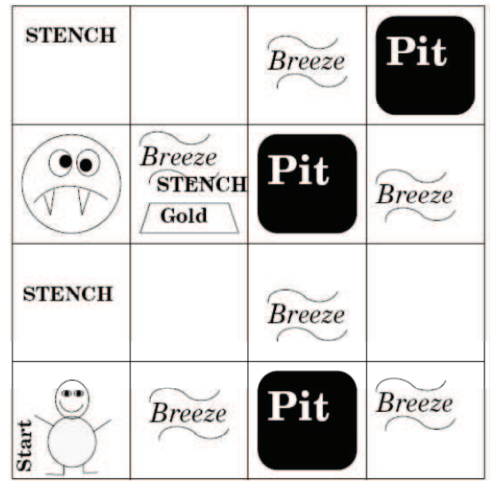
Artificial Intelligence

CS4365 --- Spring 2013

Knowledge Representation and Reasoning

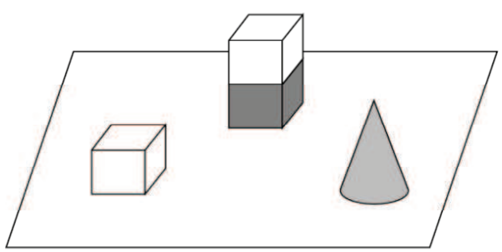
Reading: Chapters 7-9, R&N

The Wampus World



A decision-maker needs to represent knowledge of the world and reason with it in order to safely explore this world.

The Blocks World



Knowledge Representation

- Human intelligence relies on a lot of background knowledge (the more you know, the easier many tasks become / **"knowledge is power"**)
- E.g. SEND + MORE = MONEY puzzle.
- Natural language understanding
  - — Time flies like an arrow.
  - — Fruit flies like bananas.
  - — The spirit is willing but the flesh is weak. (English)
  - — The vodka is good but the meat is rotten. (Russian)
- Or: Plan a trip to L.A.

- Q. How did we encode (domain) knowledge so far?

  For search problems?

Fine for limited amounts of knowledge / well-defined domains.

Otherwise: **knowledge-based systems approach**.

Knowledge-Based Systems / Agents

Key components:

- **knowledge base**: a set of *sentences* expressed in some knowledge representation language
- **inference / reasoning mechanisms** to query what is known and to derive new information or make decisions

Natural candidate: logical language (propositional / first-order) combined with a logical inference mechanism

How close to human thought?

In any case, appears reasonable strategy for machines.

Logic as a Knowledge Representation

Three components:

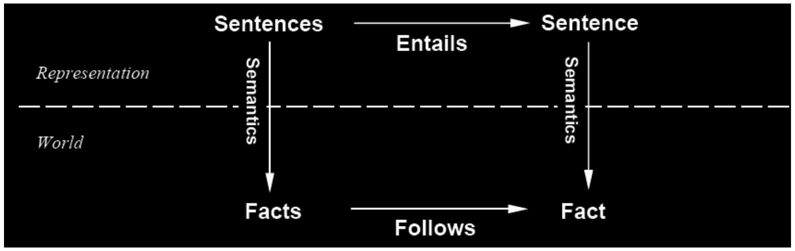
syntax: specifies which sentences can be constructed in a given formal logic

semantics: specifies what a sentence means

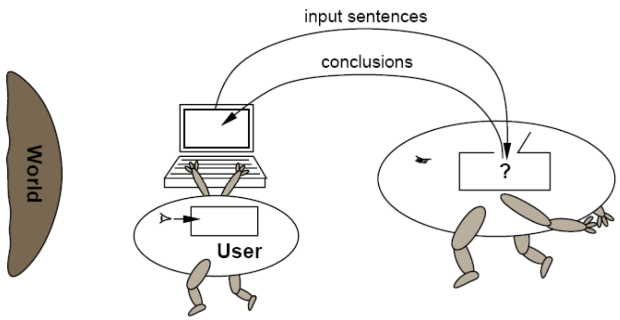
proof theory: a set of general purpose rules that allow efficient derivation of new information from the sentences in the knowledge base

To make it work, we need a **sound** and **complete** proof theory.

Connecting Sentences to the Real World



Tenuous Link to Real World



All computer has are sentences (hopefully about the world).

KR Language: Propositional Logic

Syntax: build sentences from atomic propositions, using connectives $\vee$, $\wedge$, $\neg$, $\Rightarrow$, $\Leftrightarrow$.
(and / or / not / implies / equivalence (biconditional))

E.g.: $((\neg P) \vee (Q \wedge R)) \Rightarrow S$

Semantics

Semantics specifies what something **means**.

In propositional logic, the **semantics** (i.e., meaning) of a sentence is the set of **interpretations** (i.e., truth assignments) in which the sentence evaluates to True.

Example:

The semantics of the sentence $P \vee Q \Rightarrow R$ is

- P is True , Q is True , R is True
- P is True , Q is False, R is True
- P is False , Q is True , R is True
- P is False , Q is False , R is True
- P is False , Q is False , R is False

Interpretations: The Key to Semantics

An interpretation is a logician's word for "truth assignment".

Given 3 propositional symbols *P*, *Q*, *R*, there are 8 interpretations.

Given *n* propositional symbols $P_1, P_2 \dots P_n$, there are 2^n interpretations.

In propositional logic

- an interpretation is a mapping from propositional symbols to truth values.
- the meaning of a sentence is the set of interpretations in which the sentence evaluates to True.

How to evaluate a sentence under a given interpretation?

Evaluating a sentence under interpretation I

We can evaluate a sentence using a **truth table**.

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
False	*False*	*True*	*False*	*False*	*True*	*True*
False	*True*	*True*	*False*	*True*	*True*	*False*
True	*False*	*False*	*False*	*True*	*False*	*False*
True	*True*	*False*	*True*	*True*	*True*	*True*

Note: $\Rightarrow$ somewhat counterintuitive.

What's the truth value of "5 is even implies Sam is smart"?

Validity

Some sentences are very true! For example:

(1) True (2) P => P (3) (P $\wedge$ Q) => Q

A **valid** sentence is one whose meaning includes every possible interpretation.

P	H	$P \vee H$	$(P \vee H) \wedge \neg H$	$((P \vee H) \wedge \neg H) \Rightarrow P$
False	*False*	*False*	*False*	*True*
False	*True*	*True*	*False*	*True*
True	*False*	*True*	*True*	*True*
True	*True*	*True*	*False*	*True*

The truth table shows that $((P \vee H) \wedge (\neg H)) \Rightarrow P$ is valid.

We write $\models ((P \vee H) \wedge (\neg H)) \Rightarrow P$

Satisfiability

An **unsatisfiable** sentence is one whose meaning has no interpretation (e.g., P $\wedge$ $\neg$P, False).

A **satisfiable** sentence is one whose meaning has at least one interpretation.

- A sentence must be either satisfiable or unsatisfiable but it can't be both.
- If a sentence is valid then it's satisfiable.
- If a sentence is satisfiable then it may or may not be valid.

Models

A **model** of a set of sentences (KB) is a truth assignment in which each of the KB sentences evaluates to *True*.

With more and more sentences, the models of KB start looking more and more like the "real-world".

If a sentence α holds (is *True*) in **all** models of a KB, we say that α is **entailed** by the KB.

α is of interest, because *whenever KB is true in a world* α *will also be True*.

We write $KB \models \alpha$.

Entailment Examples

KB =

- CS4365Lectures $\Rightarrow$ (TodayIsMonday $\vee$ TodayIsWednesday)
- $\neg$TodayIsWednesday
- TodayIsSaturday $\Rightarrow$ SleepLate
- Rainy $\Rightarrow$ GrassIsWet
- CS4365Lectures $\vee$ TodayIsSaturday
- $\neg$SleepLate

Then which of these are correct entailments?

KB |= $\neg$SleepLate
KB |= GrassIsWet
KB |= $\neg$SleepLate $\vee$ GrassIsWet
KB |= TodayIsMonday

Logical Inference

Problem definition:

The computer has a knowledge base KB. The user inputs a sentence. The computer tells the user whether the sentence is entailed by the knowledge base.

Given *N* propositional symbols in the system, can you invent an inference algorithm that costs $O(2^N)$ time complexity?

Humans who are doing proofs almost never use this brute-force approach. Then how to do logical inference **efficiently**?

Proof Theory

A set of purely syntactic rules for **efficiently** determining entailment.

We write: $KB \vdash \alpha$, i.e., α can be **deduced** from KB or α is **provable** from KB.

Key property:
Both in propositional and in first-order logic we have a proof theory ("calculus") such that:

$\vdash$ **and** $\models$ **are equivalent**.

Proof Theory (cont.)

If $KB \vdash \alpha$ implies $KB \models \alpha$, we say the proof theory is **sound**.

If $KB \models \alpha$ implies $KB \vdash \alpha$, we say the proof theory is **complete**.

Why so important?
Allow computer to ignore semantics and "just push symbols"!

Example Proof Theory

One rule of inference: **Modus Ponens**

From α and $\alpha \Rightarrow \beta$ it follows that β.

Semantic soundness can easily be verified (using truth table).

Axiom schemas:

(Ax. I) $\alpha \Rightarrow (\beta \Rightarrow \alpha)$

(Ax. II) $((\alpha \Rightarrow (\beta \Rightarrow \gamma)) \Rightarrow ((\alpha \Rightarrow \beta) \Rightarrow (\alpha \Rightarrow \gamma)))$

(Ax. III) $(\neg\alpha \Rightarrow \beta) \Rightarrow (\neg\alpha \Rightarrow \neg\beta) \Rightarrow \alpha$

Note: α, β, γ stand for arbitrary sentences. So, we have an infinite collection of axioms.

Now, α can be **deduced** from a set of sentences Φ iff there exists a sequence of applications of **modus ponens** that leads from Φ to α (possibly using the axioms).

One can prove that:

Modus ponens with the above axioms will generate exactly all (and only those) statements logically **entailed** by Φ.

So, we have a way of generating entailed statements *in a purely syntactic manner*!

(Sequence is called a proof. Finding it can be hard …)

Example Proof

Lemma. 1) For any α, we have $\vdash (\alpha \Rightarrow \alpha)$.

Proof.

$(\alpha \Rightarrow (\alpha \Rightarrow \alpha) \Rightarrow \alpha) \Rightarrow (\alpha \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow \alpha$, (Ax. II)

$\alpha \Rightarrow (\alpha \Rightarrow \alpha) \Rightarrow \alpha$, (Ax. I)

$(\alpha \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow \alpha$, (Modus Ponens)

$\alpha \Rightarrow \alpha \Rightarrow \alpha$, (Ax. I)

$\alpha \Rightarrow \alpha$ (Modus Ponens)

Another Example Proof

Lemma. 2) For any α and β, we have $\beta, \neg\beta \vdash \alpha$.

Proof.

$(\neg\alpha \Rightarrow \beta) \Rightarrow (\neg\alpha \Rightarrow \neg\beta) \Rightarrow \alpha$, (Ax. III)

β, (hyp.)

$\beta \Rightarrow \neg\alpha \Rightarrow \beta$, (Ax. I)

$\neg\alpha \Rightarrow \beta$, (Modus Ponens)

$(\neg\alpha \Rightarrow \neg\beta) \Rightarrow \alpha$, (Modus Ponens)

$\neg\beta$ (hyp.)

$\neg\beta \Rightarrow \neg\alpha \Rightarrow \neg\beta$, (Ax. I)

$\neg\alpha \Rightarrow \neg\beta$, (Modus Ponens)

α (Modus Ponens)

Why are lemma 1 and 2 true semantically?

I.e., $\models \alpha \Rightarrow \alpha$ and $\beta, \neg\beta \models \alpha$.

Note: **proofs** are purely **syntactic** --- machine does not need to know anything about the meaning of the sentences!

Whatever is syntactically derived will be semantically true, and we can derive everything syntactically that is semantically true.

How hard is it to find proofs?

Key Properties

We have the following properties (also for first-order logic):

For a sound and complete proof theory, the following three conditions are equivalent:

(I) $\Phi \models \alpha$

(II) $\Phi \vdash \alpha$

(III) $\Phi, \neg\alpha$ is inconsistent (i.e., can be refuted)

(I) is semantic; (II) syntactic; (III) at high-level semantic but we have a nice syntactic automatic procedure: **resolution**.

What common proof technique does III represent?

Resolution

First need canonical form: "clausal".

Conjunction of disjunctions / CNF (conjunctive normal form)

Example: $\neg(P \Rightarrow Q) \vee (R \Rightarrow P)$.

$\neg(\neg P \vee Q) \vee (\neg R \vee P)$

$(P \wedge \neg Q) \vee (\neg R \vee P)$ (de Morgan's law)

$(P \vee \neg R \vee P) \wedge (\neg Q \vee \neg R \vee P)$ (assoc. and distr. laws)

$(P \vee \neg R) \wedge (\neg Q \vee \neg R \vee P)$

$\{(P \vee \neg R), (\neg Q \vee \neg R \vee P)\}$

$\{\{P, \neg R\}, \{\neg Q, \neg R, P\}\}$ (just notation)

Given a CNF, a **single** inference rule (and no axioms) will allow us to determine inconsistency.

So, using property III (above) and resolution, we have a sound and complete proof procedure for propositional logic (which can be extended to first-order logic).

The Resolution Rule (clausal form)

From $\alpha \vee p$ and $\neg p \vee \beta$, we can derive:
$\alpha \vee \beta$ (α and β are disjunctions of literals, where a literal is a propositional variable or its negation).

Note: $\neg\alpha \Rightarrow p$ and $p \Rightarrow \beta$ gives $\neg\alpha \Rightarrow \beta$.

It's a "**chaining rule"**.

General Resolution Rule

If $(L_1 \vee L_2 \vee \ldots L_k)$ is true,
and $(\neg L_k \vee L_{k+1} \vee \ldots L_m)$ true,
then we can conclude that
$(L_1 \vee L_2 \vee \ldots L_{k-1} \vee L_{k+1} \vee \ldots \vee L_m)$ is true.

We can derive the empty clause via resolution iff the set of clauses is inconsistent.

Method relies on property III. It's **refutation complete**. Note that method does not generate theorems from scratch.
E.g., we have $P \wedge R \models (P \vee R)$, but we can't get $(P \vee R)$ from {{P}, {R}}. (Why not??)

But, given {{P},{R}} and the negation of $P \vee R$, we get the set {{P},{R},{¬P},{¬R}}. Resolving on this set gives the empty clause. Thus contradiction. Thus proof.

Algorithm: Resolution Proof

- Negate the theorem to be proved, and add the result to the list of sentences in the KB.
- Put the list of sentences into conjunctive normal form.
- Until there is no resolvable pair of clauses,
  - Find resolvable clauses and resolve them.
  - Add the results of resolution to the list of clauses.
  - If NIL (empty clause) is produced, stop and report that the (original) theorem is true.
- Report that the (original) theorem is false.

Example.

1) ARM-OK
2) $\neg$MOVES
3) ARM-OK $\wedge$ LIFTABLE $\Rightarrow$ MOVES
4) $\neg$ARM-OK $\vee$ $\neg$LIFTABLE $\vee$ MOVES
Prove: $\neg$LIFTABLE.

5) LIFTABLE (assert)
6) $\neg$ARM-OK $\vee$ MOVES (resolving 5 and 4)
7) $\neg$ARM-OK (from 6 and 2)
8) Nil (empty clause / contradiction, from 7 and 1).

Resolution Theorem Proving

Procedure may seem cumbersome, but can be easily automated. Just "smash" clauses till empty clause or no more new clauses.
Guaranteed sound and (refutation) complete.

Length of Resolution Proof

Consider Pigeon-Hole (PH) problem: Formula encodes that you cannot place n+1 pigeons in n holes (one per hole).

PH takes **exponentially** many steps (no matter in what order)!

PH hidden in many practical problems. Makes theorem proving expensive.

Pigeon-Hole Principle

$P_{i,j}$ for Pigeon i in hole j.
$P_{1,1} \vee P_{1,2} \vee P_{1,3} \dots P_{1,n}$
$P_{2,1} \vee P_{2,2} \vee P_{2,3} \dots P_{2,n}$
…
$P_{(n+1),1} \vee P_{(n+1),2} \vee P_{(n+1),3} \dots P_{(n+1),n}$
and ??

$(\neg P_{1,1} \vee \neg P_{1,2})$, $(\neg P_{1,1} \vee \neg P_{1,3})$, $(\neg P_{1,1} \vee \neg P_{1,4})$
…
$(\neg P_{1,(n\text{-}1)} \vee \neg P_{1,n})$,
$(\neg P_{2,1} \vee \neg P_{2,2})$ … $(\neg P_{2,(n\text{-}1)} \vee \neg P_{2,n})$
etc.
$(\neg P_{1,1} \vee \neg P_{2,1})$, $(\neg P_{1,1} \vee \neg P_{3,1})$, …
$(\neg P_{1,2} \vee \neg P_{2,2})$, $(\neg P_{1,2} \vee \neg P_{3,2})$, etc.

Resolution proof of inconsistency requires at least an exponential number of clauses, no matter in what order you resolve things!

A More Concise Formulation

$\forall x \exists y (x \in \text{Pigeons})(y \in \text{Holes}) IN(x, y)$
$\forall x \forall x' \forall y (IN(x, y) \wedge IN(x', y)$ … ??
$\forall x \forall y \forall y' (IN(x, y) \wedge IN(x, y')$ … ??
Pigeons = $\{p_1, p_2, \ldots, p_{n+1}\}$,
Holes = $\{h_1, h_2, \ldots, h_n\}$.

We have first-order logic with some set-theory notation.

KR Language: First-order Logic

Gives us a more concise formulation.

Essentially equivalent to propositional logic in finite domains. Often pays off to expand!

Extends propositional logic with variables, predicates, symbols, functions, and quantifiers ($\exists$, $\forall$).

Key properties from propositional case carry over: model-theoretic semantics, sound and complete proof theory, sound and refutation complete resolution.

Inference in First-Order Logic

How do we reason with first-order logic? Derive new info?

We can use **resolution** as in propositional case:

From $(\alpha \vee p) \wedge (\neg p \vee \beta)$
conclude $\alpha \vee \beta$ until you reach contradiction.

But we need some extra “tricks” to deal with **quantifiers** and **variables**.

Resolution

I **put KB in CNF (clausal) form**
all variables universally quantified
main trick: “skolemization” to remove existentials
idea: invent names for unknown objects known to exist

II **use unification** to match atomic sentences

III **apply resolution rule** to the clausal set combined with negated goal. Attempt to generate empty clause.

Converting more complicated axioms to CNF

Axiom:

$$\forall x : brick(x) \rightarrow ((\exists y : on(x, y) \wedge \neg pyramid(y)) \wedge (\neg \exists y : on(x, y) \wedge on(y, x)) \wedge (\forall y : \neg brick(y) \rightarrow \neg equal(x, y)))$$

$\neg brick(x) \vee on(x, support(x))$

$\neg brick(w) \vee \neg pyramid(support(w))$

$\neg brick(u) \vee \neg on(u, y) \vee \neg on(y, u)$

$\neg brick(v) \vee brick(z) \vee \neg equal(v, z)$

1. Eliminate implications

Substitute $\neg E_1 \vee E_2$ for $E_1 \rightarrow E_2$

$$\forall x : brick(x) \rightarrow ((\exists y : on(x, y) \wedge \neg pyramid(y)) \wedge (\neg \exists y : on(x, y) \wedge on(y, x)) \wedge (\forall y : \neg brick(y) \rightarrow \neg equal(x, y)))$$

$$\forall x : \neg brick(x) \vee ((\exists y : on(x, y) \wedge \neg pyramid(y)) \wedge (\neg \exists y : on(x, y) \wedge on(y, x)) \wedge (\forall y : \neg(\neg brick(y)) \vee \neg equal(x, y)))$$

2. Move negations down to the atomic formulas

$\neg(E_1 \wedge E_2) \Leftrightarrow (\neg E_1) \vee (\neg E_2)$
$\neg(E_1 \vee E_2) \Leftrightarrow (\neg E_1) \wedge (\neg E_2)$
$\neg(\neg E_1) \Leftrightarrow E_1$
$\neg \forall x : E_1(x) \Leftrightarrow \exists x : \neg E_1(x)$
$\neg \exists\, x : E_1(x) \Leftrightarrow \forall x : \neg E_1(x)$

$\forall x : \neg brick(x) \vee$
$((\exists y : on(x, y) \wedge \neg pyramid(y))$
$\wedge\ (\neg \exists y : on(x, y) \wedge on(y, x))$
$\wedge\ (\forall y : \neg(\neg brick(y)) \vee \neg equal(x, y)))$

3. Eliminate existential quantifiers

Skolemization

Harder cases:
$\forall x : \exists y : father(y, x)$ becomes $\forall x : father(S1(x), x)$

There is one argument for each universally quantified variable whose scope contains the Skolem function.

Easy case:
$\exists x : President(x)$ becomes $President(S2)$
$\forall x : \neg brick(x) \vee ((\exists y : on(x, y) \wedge \neg pyramid(y)) \wedge \ldots$

4. Rename variables as necessary

We want no two variables of the same name.

$$\begin{aligned}\forall x : \neg brick(x) \vee ((on(x, S1(x)) \wedge \neg pyramid(S1(x)))\\ \wedge\ (\forall y : (\neg on(x, y) \vee \neg on(y, x)))\\ \wedge\ (\forall y : (brick(y) \vee \neg equal(x, y))))\end{aligned}$$

$$\begin{aligned}\forall x : \neg brick(x) \vee ((on(x, S1(x)) \wedge \neg pyramid(S1(x)))\\ \wedge\ (\forall y : (\neg on(x, y) \vee \neg on(y, x)))\\ \wedge\ (\forall z : (brick(z) \vee \neg equal(x, z))))\end{aligned}$$

5. Move the universal quantifiers to the left

This works because each quantifier uses a unique variable name.

$$\begin{aligned}\forall x : \neg brick(x) \vee ((on(x, S1(x)) \wedge \neg pyramid(S1(x)))\\ \wedge\ (\forall y : (\neg on(x, y) \vee \neg on(y, x)))\\ \wedge\ (\forall z : (brick(z) \vee \neg equal(x, z))))\end{aligned}$$

$$\begin{aligned}\forall x \forall y \forall z : \neg brick(x) \vee ((on(x, S1(x)) \wedge \neg pyramid(S1(x)))\\ \wedge\ (\neg on(x, y) \vee \neg on(y, x))\\ \wedge\ (brick(z) \vee \neg equal(x, z)))\end{aligned}$$

6. Move disjunctions down to the literals

$E_1 \vee (E_2 \wedge E_3) \Leftrightarrow (E_1 \vee E_2) \wedge (E_1 \vee E_3)$

$$\begin{aligned} \forall x \forall y \forall z : (&\neg brick(x) \vee (on(x, S1(x)) \wedge \neg pyramid(S1(x)))) \\ &\wedge (\neg brick(x) \vee \neg on(x, y) \vee \neg on(y, x)) \\ &\wedge (\neg brick(x) \vee brick(z) \vee \neg equal(x, z)) \end{aligned}$$

$$\begin{aligned} \forall x \forall y \forall z : (&\neg brick(x) \vee on(x, S1(x))) \\ &\wedge (\neg brick(x) \vee \neg pyramid(S1(x))) \\ &\wedge (\neg brick(x) \vee \neg on(x, y) \vee \neg on(y, x)) \\ &\wedge (\neg brick(x) \vee brick(z) \vee \neg equal(x, z)) \end{aligned}$$

7. Eliminate the conjunctions

$$\begin{aligned} \forall x \forall y \forall z : (&\neg brick(x) \vee on(x, S1(x))) \\ &\wedge (\neg brick(x) \vee \neg pyramid(S1(x))) \\ &\wedge (\neg brick(x) \vee \neg on(x, y) \vee \neg on(y, x)) \\ &\wedge (\neg brick(x) \vee brick(z) \vee \neg equal(x, z)) \end{aligned}$$

$\forall x : \neg brick(x) \vee on(x, S1(x))$

$\forall x : \neg brick(x) \vee \neg pyramid(S1(x))$

$\forall x \forall y : \neg brick(x) \vee \neg on(x, y) \vee \neg on(y, x)$

$\forall x \forall z : \neg brick(x) \vee brick(z) \vee \neg equal(x, z)$

8. Rename all variables, as necessary, so no two have the same name

$\forall x : \neg brick(x) \vee on(x, S1(x))$

$\forall x : \neg brick(x) \vee \neg pyramid(S1(x))$

$\forall x \forall y : \neg brick(x) \vee \neg on(x, y) \vee \neg on(y, x)$

$\forall x \forall z : \neg brick(x) \vee brick(z) \vee \neg equal(x, z)$

$\forall x : \neg brick(x) \vee on(x, S1(x))$

$\forall w : \neg brick(w) \vee \neg pyramid(S1(w))$

$\forall u \forall y : \neg brick(u) \vee \neg on(u, y) \vee \neg on(y, u)$

$\forall v \forall z : \neg brick(v) \vee brick(z) \vee \neg equal(v, z)$

9. Eliminate the universal quantifiers

$\neg brick(x) \vee on(x, S1(x))$

$\neg brick(w) \vee \neg pyramid(S1(w))$

$\neg brick(u) \vee \neg on(u, y) \vee \neg on(y, u)$

$\neg brick(v) \vee brick(z) \vee \neg equal(v, z)$

Algorithm: Putting Axioms into Clausal Form

- Eliminate the implications.
- Move the negations down to the atomic formulas.
- Eliminate the existential quantifiers.
- Rename the variables, if necessary.
- Move the universal quantifiers to the left.
- Move the disjunctions down to the literals.
- Eliminate the conjunctions.
- Rename the variables, if necessary.
- Eliminate the universal quantifiers.

Unification

`UNIFY(P,Q)` takes two atomic sentences `P` and `Q` and returns a substitution that makes `P` and `Q` **look the same.**

Rules for substitutions:

- Can replace a variable by a constant.
- Can replace a variable by a variable.
- Can replace a variable by a function expression, as long as the function expression does not contain the variable.

Unifier: a substitution that makes two clauses resolvable.
*v*1/*C*; *v*2/*v*3; *v*4/*f*(...)

Unification -- Purpose

Given:

Knows(*John, x*) → *Hates*(*John, x*)
Knows(*John, Jim*)

Derive:

Hates(*John, Jim*)

Need **unifier** *{x/Jim}* for resolution to work.
(simplest case)

¬*Knows*(*John, x*) ∨ *Hates*(*John, x*)
Knows(*John, Jim*)

How do we resolve? First, match them.
Solution:
UNIFY(*Knows*(*John, x*),*Knows*(*John, Jim*)) = {*x/Jim*}
Gives
¬*Knows*(*John, Jim*) ∨ *Hates*(*John, Jim*) and
Knows(*John, Jim*)

Conclude by resolution
Hates(*John, Jim*)

Unification (example)

one rule:

$Knows(John, x) \rightarrow Hates(John, x)$

facts:

$Knows(John, Jim)$
$Knows(y, Leo)$
$Knows(z, Mother(z))$
$Knows(x, Jane)$

Who does John hate?

UNIFY($Knows(John,x)$,$Knows(John,Jim)$) = $\{x/Jim\}$

UNIFY($Knows(John,x)$,$Knows(y,Leo)$) = $\{x/Leo, y/John\}$

UNIFY($Knows(John,x)$,$Knows(z,Mother(z))$) = $\{z/John, x/Mother(John)\}$

UNIFY($Knows(John,x)$,$Knows(x, Jane)$) = *fail*

Most General Unifier

In cases where there is more than one substitution choose the one that makes the least commitment (most general) about the bindings.

UNIFY($Knows(John,x)$,$Knows(y, z)$)
= $\{y/John, x/z\}$
or $\{y/John, x/z, z/Freda\}$
or $\{y/John, x/John, z/John\}$
or ....

See R&N for general unification algorithm.

Completeness

Resolution with **unification** applied to **clausal form** is a complete inference procedure.

In practice, still a significant search problem!
Many different search strategies: **resolution strategies**.

Strategies for Selecting Clauses

unit-preference strategy: Give preference to resolutions involving the clauses with the smallest number of literals.

set-of-support strategy: Try to resolve with the negated theorem or a clause generated by resolution from that clause.

subsumption: Eliminates all sentences that are subsumed (i.e., more specific than) an existing sentence in the KB.