Goal

Deductive reasoning in language as close as possible to full FOL

\[ \neg, \land, \lor, \exists, \forall \]

Knowledge Level:

given KB, \( \alpha \), determine if KB \( \models \alpha \).

or

given an open \( \alpha(x_1, x_2, \ldots, x_n) \), find \( t_1, t_2, \ldots, t_n \)
such that KB \( \models \alpha(t_1, t_2, \ldots, t_n) \)

When KB is finite \( \{ \alpha_1, \alpha_2, \ldots, \alpha_k \} \)

KB \( \models \alpha \)

iff \( \models [(\alpha_1 \land \alpha_2 \land \ldots \land \alpha_k) \supset \alpha] \)

iff KB \( \cup \{ \neg \alpha \} \) is unsatisfiable

iff KB \( \cup \{ \neg \alpha \} \models \) FALSE

So want a procedure to test for validity, or satisfiability, or for entailing FALSE.

Will now consider such a procedure

first: without quantifiers

Clausal Representation

Formula = set of clauses

Clause = set of literals

Literal = atomic sentence or its negation

positive literal and negative literal

Notation:

If \( l \) is a literal, then \( \neg l \) is its complement

\[ p \Rightarrow \neg p \quad \neg p \Rightarrow p \]

To distinguish clauses from formulas:

- [ and ] for clauses: \[ [p, \neg r, s] \]
- { and } for formulas: \[ \{[p, r, s], [p, \neg r, s], [\neg p] \} \]

\[ () \] is the empty clause

\[ {} \] is the empty formula

So {} is different from [][]

Interpretation:

Formula understood as conjunction of clauses

Clause understood as disjunction of literals

Literals understood normally

So:

\[ \{[p, \neg q], [r], [s] \} \] is representation of \( ((p \lor \neg q) \land r \land s) \)

[] is a representation of FALSE

{} is a representation of TRUE
Every propositional wff $\alpha$ can be converted into a formula $\alpha'$ in Conjunctive Normal Form (CNF) in such a way that $\models \alpha \equiv \alpha'$.

1. eliminate $\supset$ and $\equiv$ using $(\alpha \supset \beta) \equiv (\neg \alpha \lor \beta)$ etc.
2. push $\neg$ inward
   using $\neg (\alpha \land \beta) \equiv (\neg \alpha \lor \neg \beta)$ etc.
3. distribute $\lor$ over $\land$
   using $((\alpha \land \beta) \lor \gamma) \equiv ((\alpha \lor \gamma) \land (\beta \lor \gamma))$
4. collect terms
   using $(\alpha \lor \alpha) \equiv \alpha$ etc.

Result is a conjunction of disjunction of literals.

We can identify CNF wffs with clausal formulas

\[
(p \lor \neg q \land s \lor \neg r) \equiv \{[p, \neg q, r], [s, \neg r]\}
\]

So: given a finite KB and $\alpha$, to find out if

$KB \models \alpha$, it will be sufficient to

1. put $(KB \land \neg \alpha)$ into CNF, as above
2. determine the satisfiability of clauses

\[\text{Resolution rule of inference}\]

Given two clauses, infer a new clause:

From clause $\langle p \rangle \cup C_1$, and $\langle \neg p \rangle \cup C_2$, infer clause $C_1 \cup C_2$.

$C_1 \cup C_2$ is called a \textit{resolvent} of input clauses with respect to $p$.

Example:

From clauses $[w, p, q]$ and $[w, s, \neg p]$, have $[w, q, s]$ as resolvent wrt $p$.

Special Case:

$[p]$ and $[\neg p]$ resolve to $[]$

$C_1$ and $C_2$ are empty

A \textit{derivation} of a clause $c$ from a set $S$ of clauses is a sequence $c_1, c_2, \ldots, c_n$ of clauses, where the last clause $c_n = c$, and for each $c_i$, either

1. $c_i \in S$, or
2. $c_i$ is a resolvent of two earlier clauses in the derivation

Write: $S \models c$ if there is a derivation
Resolution is a symbol-level rule of inference, but has a connection to knowledge-level logical interpretations.

Resolvent is entailed by input clauses.

Suppose \( I \models (p \lor \alpha) \) and \( I \models (\neg p \lor \beta) \)

Case 1: \( I \notmodels p \)
then \( I \models \beta \), so \( I \models (\alpha \lor \beta) \).

Case 2: \( I \models p \)
then \( I \models \alpha \), so \( I \models (\alpha \lor \beta) \).

Either way, \( I \models (\alpha \lor \beta) \).

So: \( \{(p \lor \alpha), (\neg p \lor \beta)\} \models (\alpha \lor \beta) \).

Special case:

\([p]\) and \([\neg p]\) resolve to [],
so \( \{[p], [\neg p]\} \models \text{FALSE} \)
that is: \( \{[p], [\neg p]\} \) is unsatisfiable.

---

Derivations and entailment

Can extend the previous argument to derivations:

If \( S \vdash c \) then \( S \models c \)
Proof: by induction on the length of the derivation.
Show (by looking at the two cases) that \( S \models c \).

But the converse does not hold in general

Can have \( S \models c \) without having \( S \vdash c \).

Example: \( \{[\neg p]\} \models [\neg p, \neg q] \)
i.e. \( \neg \neg \neg q \models (\neg p \lor \neg q) \)

but no derivation

However, ...

Resolution is sound and complete for []!

Theorem: \( S \vdash [] \) if and only if \( S \models [] \)
Result will carry over to quantified clauses (later)

So for any set \( S \) of clauses:

\( S \) is unsatisfiable if and only if \( S \vdash [] \).

Provides method for determining satisfiability:
Search all derivations to see if [] is produced
Also provides method for determining all entailments
A procedure for entailment

To determine if $\text{KB} \models \alpha$
- put $\text{KB}, \neg \alpha$ into CNF to get $S$, as before
- check if $S \vdash \emptyset$.

If $\text{KB} = \emptyset$, then we are testing the validity of $\alpha$.

Non-deterministic procedure
1. Check if $\emptyset$ is in $S$.
   If yes, then return UNSATISFIABLE
2. Check if there are two clauses $c_1$ and $c_2$ in $S$ such that they resolve to produce a $c_j$ not already in $S$.
   If no, then return SATISFIABLE
3. Add $c_j$ to $S$ and go to 1.

Note: need only convert $\text{KB}$ to CNF once
- can handle multiple queries with same $\text{KB}$
- after addition of new fact $\alpha$, can simply add new clauses $\alpha'$ to $\text{KB}$

Example 1

KB:
- FirstGrade
- FirstGrade $\supset$ Child
- Child $\land$ Male $\supset$ Boy
- Kindergarten $\supset$ Child
- Child $\land$ Female $\supset$ Girl
- Female

Show that $\text{KB} \models \text{Girl}$

Derivation has 9 clauses, 4 new
Example 2

KB:
(Rain ∨ Sun)
(Sun ⊃ Mail)
((Rain ∨ Sleet) ⊃ Mail)

Show KB |= Mail

Similarly KB |= Rain

Can enumerate all clauses given ¬Rain
and [] will not be generated

Quantifiers

Clausal form as before, but atom is
\( P(t_1, t_2, ..., t_n) \), where \( t_i \) may contain variables

Interpretation as before, but variables are understood universally

Example: \{[P(x), ¬ R(a,f(b,x))], [Q(x,y)]\}

interpreted as
\[ \forall x \forall y \{ [R(a,f(b,x)) \supset P(x)] \land Q(x,y) \} \]

Substitutions: \( \theta = \{ v_1/t_1, v_2/t_2, ..., v_n/t_n \} \)

Notation: If \( l \) is a literal and \( \theta \) is a substitution, then \( l\theta \) is the result of the substitution
(and similarly, \( c\theta \) where \( c \) is a clause)

Example: \( \theta = \{ x/a, y/g(x,b,z) \} \)

\[ P(x,z,f(x,y)) \theta = P(a,z,f(a,g(x,b,z))) \]

A literal is ground if it contains no variables.
A literal \( l \) is an instance of \( l' \),
if for some \( \theta \), \( l = l'\theta \).
Generalizing CNF

Resolution will generalize to handling variables
But how to convert wffs to CNF?
Need three additional steps: 

1. eliminate $\supset$ and $\equiv$
2. push $\neg$ inward
   using also $\neg \forall \alpha$ & $\exists x \alpha$, etc.

3. standardize variables: each quantifier gets its own variable
   e.g. $\exists x P(x) \land Q(z) \land \exists y P(y)$
   where $z$ is a new variable

4. eliminate all existentials
   (discussed later)

5. move universals to the front
   using $\forall x \alpha \land \beta \land \forall y \beta$ (where $\beta$ does not use $x$)

6. distribute $\lor$ over $\land$

7. collect terms

Get universally quantified conjunction of disjunction of literals
then drop the quantifiers...

First-order resolution

Main idea:
   a literal (with variables) stands for all its instances;
   will allow all such inferences
So given:
   $[P(x,a), \neg Q(x)]$ and $[\neg P(b,y), \neg R(b,f(y))],$
   want to infer: $[\neg Q(b), \neg R(b,f(a))]$
   since $[P(x,a), \neg Q(x)]$ has $[P(b,a), \neg Q(b)]$ and
   $[\neg P(b,y), \neg R(b,f(y))]$ has $[\neg P(b,a), \neg R(b,f(a))]$

Resolution:
   Given clauses: $l_1 \cup C_1$ and $\neg l_2 \cup C_2$
   Rename variables, so that distinct in two clauses.
   For any $\theta$ such that $l_1 \theta = l_2$, can infer $(C_1 \cup C_2) \theta$
   e.g. below

   We say that $l_1$ unifies with $l_2$ and
   that $\theta$ is a unifier of the two literals

Resolution derivation: as before still ignoring $=$

Theorem: $S \models []$ iff $S \models []$
iff $S$ is unsatisfiable
### Example 3

**KB:**

\[
\forall x \text{ GradStudent}(x) \supset \text{Student}(x) \\
\forall x \text{ Student}(x) \supset \text{HardWorker}(x) \\
\text{GradStudent}(\text{sue})
\]

**Q:** \text{HardWorker}(\text{sue})

\[
\neg \text{HardWorker}(\text{sue}) \\
\neg \text{Student}(\text{sue}), \text{HardWorker}(\text{sue}) \\
\neg \text{GradStudent}(\text{sue}), \text{Student}(\text{sue}) \\
\neg \text{GradStudent}(\text{sue})
\]

Can label each step with the unifier

---

### The 3 block example

**KB = \{\text{On}(a,b), \text{On}(b,c), \text{Green}(a), \neg \text{Green}(c)\}**

already in CNF

**Q = \exists x \exists y [\text{On}(x,y) \land \text{Green}(x) \land \neg \text{Green}(y)]**

Note: \neg Q has no existentials to eliminate

yields \[
\neg \text{On}(x,y), \neg \text{Green}(x), \text{Green}(y)\]

in CNF

\[
\neg \text{Green}(b), \text{Green}(c) \\
\neg \text{Green}(c) \\
\neg \text{Green}(a), \text{Green}(b) \\
\text{Green}(b)
\]

Note: Need to use \text{On}(x,y) twice, for 2 cases
### Arithmetic

**KB:**

\[
\begin{align*}
&\text{Plus}(\text{zero}, x, x) \\
&\text{Plus}(x, y, z) \implies \text{Plus}(\text{succ}(x), y, \text{succ}(z))
\end{align*}
\]

**Q:** \(\exists u \text{ Plus}(2, 3, u)\)

where for readability, we use

0 for zero,

3 for \(\text{succ}(\text{succ}(\text{succ}(\text{zero}))\) etc.

Can find the answer

in the derivation

\(u/3\), \(w/3\)

Can derive \(\text{Plus}(2, 3, 5)\)

### Answer predicates

In full FOL, have possibility of deriving \(\exists x P(x)\)

without being able to derive \(P(t)\) for any \(t\).

e.g. the three-blocks problem

\[
\exists x \exists y [\text{On}(x, y) \land \text{Green}(x) \land \neg \text{Green}(y)]
\]

but cannot derive which block is which

**Solution:** answer-extraction process

- replace query \(\exists x P(x)\) by \(\exists x [P(x) \land \neg A(x)]\)

  where \(A\) is a new predicate symbol called the answer predicate

- instead of deriving \([\_]\), derive any clause containing just the answer predicate

- can always convert a derivation of \([\_]\)

**Example KB:**

\{\text{Student(john), Student(jane), Happy(john)}\}

**Q:** \(\exists x [\text{Student}(x) \land \text{Happy}(x)]\)

\[
\begin{align*}
&\neg \text{Student}(\text{x/john}) \\
&\text{Student}(\text{jane}) \\
&\neg \text{Student}(\text{jane}) \\
&\text{Happy}(\text{jane}) \\
&\neg \text{Happy}(\text{jane}) \\
&\text{A}(\text{jane})
\end{align*}
\]

An answer is: John
Disjunctive answers

Example KB:

\{\text{Student(john)}, \text{Student(jane)}, \\
\text{Happy(john) } \lor \text{Happy(jane)}\}

Q:  \(\exists x [\text{Student(x)} \land \text{Happy(x)}]\)

\[ \downarrow \]

\[\text{Student(jane)} \]

\[\text{Student(john)} \]

\[\text{[Happy(john), Happy(jane)]} \]

\[\text{[Happy(john), A(jane)]} \]

\[\text{[Happy(john), A(jane)]} \]

\[\text{[A(jane), A(john)]} \]

\[\text{[¬Student(\(\alpha\)), ¬Happy(\(\alpha\)), A(\(\alpha\))]} \]

\[\text{[¬Student(john), ¬Happy(john), A(john)]} \]

\[\text{[¬Student(jane), ¬Happy(jane), A(jane)]} \]

\[\text{[¬Student(\(\alpha\)), ¬Happy(\(\alpha\)), A(\(\alpha\))]} \]

An answer is: either Jane or John

Note:

- can have variables in answer
- need to watch for Skolem symbols...

Skolemization

So far, converting wff to CNF ignored existentials

e.g.  \(\exists x \forall y \exists z P(x, y, z)\)

Idea: names for individuals claimed to exist, called Skolem constant and function symbols

there exists an \(x\), call it \(a\)

for each \(y\), there is a \(z\), call it \(f(y)\)

get \(\forall y P(a, y, f(y))\)

In general:

\[\forall x_1(...\forall x_n(...\exists y [... y ...] ...))...\]

is replaced by

\[\forall x_1(...\forall x_n(...\forall x_m(...f(x_1, x_2,..., x_n) [... y ...] ...))...\]

where \(f\) is a new function symbol that appears nowhere else

Skolemization does not preserve equivalence

e.g. \(\not\models \exists x P(x) \equiv P(a)\)

But it does preserve satisfiability

\(\alpha\) is satisfiable iff \(\alpha'\) is satisfiable

where \(\alpha'\) is the result of skolemization

Sufficient for resolution!
Variable dependence

Show that $\exists x \forall y R(x,y) \models \forall y \exists x R(x,y)$

show $\{\exists x \forall y R(x,y), \neg \forall y \exists x R(x,y)\}$ unsatisfiable

$\exists x \forall y R(x,y) \models \forall y R(a,y)$

$\neg \forall x \exists y R(x,y) \models \exists y \forall x \neg R(x,y)$

then $\{[R(a,y)], [\neg R(x,b)]\} \models []$ with $\{x/a, y/b\}$.

Show that $\forall y \exists x R(x,y) \models \neg \exists x \forall y R(x,y)$

show $\{\forall y \exists x R(x,y), \neg \exists x \forall y R(x,y)\}$ satisfiable

$\forall y \exists x R(x,y) \models \forall y R(f(y),y)$

$\neg \exists x \forall y R(x,y) \models \forall x \exists y \neg R(x,y) \models \forall x \neg R(x,g(x))$

then get $\{[R(f(y),y)], [\neg R(x,g(x))]\}$

where the two literals do not unify

Note:

important to get dependence of variables correct

$R(f(y),y)$ vs. $R(a,y)$ in the above

first argument depends on second one here

A problem

KB: LessThan(succ(x),y) ⊃ LessThan(x,y)
Q: LessThan(zero,zero)

Should fail since KB $\not\models$ Q

[LessThan(x,y), ¬LessThan(succ(x),y)]

[¬LessThan(0,0)]

x0, y0

[¬LessThan(1,0)]

x1, y0

[¬LessThan(2,0)]

x2, y0

... Infinite branch of resolvents

cannot use a simple depth-first procedure to search for []
Undecidability

Is there a way to detect when this happens?

No! FOL is very powerful
  can be used as a full programming language
  just as there is no way to detect in general when
  a program is looping

There can be no procedure that does this:
  \[
  \text{Proc}[\text{Clauses}] =
  \begin{align*}
  &\text{If } \text{Clauses} \text{ are unsatisfiable} \\
  &\quad \text{then return YES} \\
  &\quad \text{else return NO}
  \end{align*}
  \]

However: Resolution is complete
  some branch will contain [], for unsat clauses

So breadth-first search guaranteed to find []
  search may not terminate on satisfiable clauses

Overly specific unifiers

In general, no way to guarantee efficiency, or
  even termination
    later: put control into users’ hands

one major way:
  reduce redundancy in search, by keeping search
  as general as possible

Example
  \[
  \ldots, P(g(x),f(x),z) \quad [\neg P(y,f(w),a), \ldots
  \]

unified by
  \[
  \theta_1 = \{ x/b, y/g(b), z/a, w/b \}
  \]
  gives
  \[
  P(g(b),f(b),a)
  \]
  and by
  \[
  \theta_2 = \{ x/f(z), y/g(f(z)), z/a, w/f(z) \}
  \]
  gives
  \[
  P(g(f(z)),f(f(z)),a).
  \]

Might not be able to derive [] from clauses
  having overly specific substitutions
    wastes time in search!
Most general unifiers

\( \theta \) is a most general unifier of literals \( l_1 \) and \( l_2 \) iff

1. \( \theta \) unifies \( l_1 \) and \( l_2 \)
2. for any other unifier \( \theta' \), there is a another substitution \( \theta^* \) such that \( \theta' = \theta \theta^* \)

Note: composition \( \theta \theta^* \) requires applying \( \theta^* \) to terms in \( \theta \)

for previous example, an MGU is

\[ \theta = \{ x/w, y/g(w), z/a \} \]

for which

\[ \theta_1 = \theta\{ w/b \} \]
\[ \theta_2 = \theta\{ w/f(z) \} \]

Theorem: Can limit search to MGUs only without loss of completeness (with certain caveats)

Computing an MGU, given a set of lits \( \{ l_i \} \)

1. Start with \( \theta = \{} \).
2. If all the \( l_i \theta \) are identical, then done; otherwise, get disagreement set, \( DS \)

\( e.g. \) \( P(a,f(a,g(z),... \ P(a,f(a,u,... \)

disagreement set, \( DS = \{ a, g(z) \} \)

3. Find a variable \( v \in DS \), and a term \( t \in DS \) not containing \( v \). If not, fail.
4. \( \theta = \theta\{v/t\} \)
5. Go to 2

Note: there is a better linear algorithm

Herbrand Theorem

Some 1st-order cases can be handled by converting them to a propositional form

Given a set of clauses \( S \)

- the Herbrand universe of \( S \) is the set of all terms formed using only the function symbols (and constants, at least one) in \( S \)
  - for example, if \( S \) uses (unary) \( f \), and \( c, d \),
    \( U = \{ c, d, f(c), f(d), f(f(c)), f(f(d)), f(f(f(c))) \ldots \} \)
- the Herbrand base of \( S \) is
  \( \{ c \theta \mid c \in S \text{ and } \theta \text{ replaces the variables in } c \text{ by terms from the Herbrand universe} \} \)

Theorem: \( S \) is satisfiable iff Herbrand base is

(anylips to Horn clauses also)

Herbrand base has no variables, and so is essentially propositional, though usually infinite

- finite, when Herbrand universe is finite can use propositional methods (guaranteed to terminate)
- sometimes other "type" restrictions can be used to keep the Herbrand base finite include \( f(t) \) only if \( t \) is the correct type
Resolution is difficult!

First-order resolution is not guaranteed to terminate.

What can be said about the propositional case?

Recently shown by Haken that there are unsatisfiable clauses \( \{c_1, c_2, \ldots, c_n\} \) such that the shortest derivation of \( \emptyset \) contains on the order of \( 2^n \) clauses.

Even if we could always find a derivation immediately, the most clever search procedure will still require exponential time on some problems.

Problem just with resolution?

Probably not.

Determining if set of clauses is satisfiable shown by Cook to be NP-complete:

- no easier than an extremely large variety of computational tasks
- any search task where what is searched for can be verified in polynomial time can be recast as a satisfiability problem
  - satisfiability
  - does graph of cities allow for a full tour of size \( k \) miles?
  - can N queens be put on an N-N chessboard all safely?
  - ...

Satisfiability is strongly believed by most people to be unsolvable in polynomial time.

Implications for KR

Problem: want to produce entailments of KB as needed for immediate action

- full theorem-proving may be too difficult for KR!
- need to consider other options ...
  - giving control to user
    - procedural representations (later)
  - less expressive languages
    - e.g. Horn clauses (and a major theme later)

In some applications, it is reasonable to wait

- e.g. mathematical theorem proving, where we only care about specific formula

Best to hope for in general: reduce redundancy

- refinements to resolution to improve search

Main example: MGU, as before

- but many other possibilities
  - need to be careful to preserve completeness

ATP: automated theorem proving

- area that studies strategies for proving difficult theorems
- main application: mathematics,
  - but relevance also to KR
Strategies

1. Clause elimination
   - pure clause
     contains literal $l$ such that $\neg l$ does not appear in any other clause
     clause cannot lead to $[]$
   - tautology
     clause with a literal and its negation
     any path to $[]$ can bypass tautology
   - subsumed clause
     a clause such that one with a subset of its literals is already present
     path to $[]$ need only pass through short clause
     can be generalized to allow substitutions

2. Ordering strategies
   - many possible ways to order search, but best and simplest is
   - unit preference
     prefer to resolve unit clauses first
     Why? Given unit clause and another clause, resolvent is a smaller one $\notin []$

Strategies 2

3. Set of support
   - KB is usually satisfiable, so not very useful to resolve among clauses with only ancestors in KB
   - contradiction arises from interaction with $\neg Q$
   - always resolve with at least one clause that has an ancestor in $\neg Q$
   - preserves completeness (sometimes)

4. Connection graph
   - pre-compute all possible unifications
   - build a graph with edges between any two unifiable literals of opposite polarity
   - label edge with MGU

   Resolution procedure:
   - repeatedly:
     - select link
     - compute resolvent
     - inherit links from parents after substitution

   Resolution as search:
   - find sequence of links $L_1, L_2, \ldots$ producing $[]$
5. Special treatment for equality

Instead of using axioms for equality,
reLexity, symmetry, transitivity,
substitution of equals for equals.
Use new inference rule: paramodulation.

From \{ (t = s) \} \cup C_1 and \{ P(\ldots r' \ldots) \} \cup C_2
where \theta = r' \theta

Infer \{ P(\ldots s \ldots) \} \theta \cup C_2 \theta.
Collapses many resolution steps into one.
See also: theory resolution (later).

\[ \text{[father(john) = bill]} \quad \text{[Married(father(x), mother(x))]}
\]
\[ \text{[Married(bill, mother(john))]}
\]

6. Sorted logic

Terms get sorts:
\( x : \text{Male} \quad \text{mother} : \text{Person} \rightarrow \text{Female} \)

Keep taxonomy of sorts.
Refuse to unify \( P(x) \) with \( P(t) \) unless sorts are compatible.
Assumes only “meaningful” paths will lead to []

Finally...

7. Directional connectives

Given \([\neg p, q]\), can interpret as either

From \( p \), infer \( q \) \hspace{1cm} \text{(forward)}
To prove \( q \), prove \( p \) \hspace{1cm} \text{(backward)}

Procedural reading of \( \supset \)

In 1st case:
Would only resolve \([\neg p, q]\) with \([p, \ldots]\)
Producing \([q, \ldots]\)

In 2nd case:
Would only resolve \([\neg p, q]\) with \([\neg q, \ldots]\)
Producing \([\neg p, \ldots]\)

Intended application:

Forward: Battleship(\( x \)) \supset Gray(\( x \))

Do not want to try to prove something is gray by proving it is a battleship.

Backward: Human(\( x \)) \supset Has(\( x, \text{spleen} \))

Do not want to conclude from someone being human, that she has each property.

The basis for the procedural representations.