COMP4418: Knowledge Representation and Reasoning
Resolution

Maurice Pagnucco
School of Computer Science and Engineering
COMP4418, Week 2
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 \( \models [\{\alpha_1 \land \alpha_2 \land \ldots \land \alpha_k\} \rightarrow \alpha] \)
- iff KB \( \cup \{\neg\alpha\} \) is unsatisfiable
- iff KB \( \cup \{\neg\alpha\} \models \text{FALSE} \)

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

Formula = set of clauses
Clause = set of literals
Literal = atomic sentence or it's negation
  positive literal and negative literal
  positive predicate and negative predicate in FOL
Notation:
  • If $p$ is a literal, then $\bar{p}$ is its complement
    \[ \bar{p} \Rightarrow \neg p \quad \neg p \Rightarrow p \]
  • To distinguish clauses from formulas:
    ◦ [ and ] for clauses: $[p, \neg r, s]$
    ◦ { and } for formulas: $\{[p, \neg r, s], [p, r, s], \neg p\}$
      [] is the empty clause; {} is the empty formula
      So {} is different from {[}]!

Interpretation:
  • Formula understood as conjunction of clauses
  • Clause understood as disjunction of clauses
  • Literals understood normally

So:
  • $\{[p, \neg q], [r], s\}$ is a representation of $((p \lor \neg q) \land r \land s)$
  • [] is a representation of FALSE
  • {} is a representation of TRUE
Resolution Rule of Inference

Given two clauses, infer a new clause:
From clause \( \{p\} \cup C_1 \)
and \( \{\neg p\} \cup C_2 \),
infer clause \( C_1 \cup C_2 \).
\( C_1 \cup C_2 \) is called a 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 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: \( S \vdash c \) if there is a derivation
Resolution Rule of Inference

• **Generalised Resolution Rule:**
  
  For clauses $\chi \lor \Phi$ and $\neg \Psi \lor \zeta$

  $\chi \lor \Phi$  
  $\neg \Psi \lor \zeta$

  $(\chi \lor \zeta).\theta$

• Where $\theta$ is a unifier for atomic formulae $\Phi$ and $\Psi$

• $\chi \lor \zeta$ is known as the *resolvent*
Rationale

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: \( I \models p \)
then \( I \models \beta \), so \( I \models (\alpha \lor \beta) \).
Case 2: \( I \not\models p \)
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 \( S \models c_i \).

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., \( \neg p \models (\neg p \lor \neg q) \)

but no derivation

However, . . .

Resolution is sound and complete for \( \Box \) !

Theorem: \( S \vdash \Box \) iff \( S \models \Box \)

Result will carry over to quantified clauses (later)

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

\( S \) is unsatisfiable iff \( S \vdash \Box \).

Provides method for determining satisfiability:

Search all derivations to see if \( \Box \) is produced

Also provides method for determining all entailments

B&L (2005)
Example

KB:
\[ \forall x \text{GradStudent}(x) \rightarrow \text{Student}(x) \]
\[ \forall x \text{Student}(x) \rightarrow \text{HardWorker}(x) \]
GradStudent(sue)
Q: HardWorker(sue)

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

Can label each step with the unifier

B&L (2005)
The 3 block example

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

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
Arithmetic

KB:
Plus(zero, x, x)
Plus(x, y, z) → Plus(succ(x), y, succ(z))

Q:  ∃u Plus(2, 3, u)
where for readability, we use
  0 for zero,
  3 for succ(succ(succ(zero))) etc.

Can find the answer in the derivation
  u/succ(succ(3))
i.e. u/5
Can derive 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 $t$
eq x 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: \{Student(john), Student(jane), Happy(john)\}

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

---

B&L (2005)
Disjunctive answers

Example KB: {Student(john), Student(jane), [Happy(john) ∨ Happy(jane)]}

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

Note:
- can have variables in answer
- need to watch for Skolem symbols . . .
A Problem

KB: \( \text{LessThan}(\text{succ}(x), y) \rightarrow \text{LessThan}(x, y) \)
Q: \( \text{LessThan}(\text{zero}, \text{zero}) \)
    Should fail since KB \( \not\models Q \)

\[
\begin{align*}
\text{[LessThan}(x, y), \neg\text{LessThan}(\text{succ}(x), y)] & \\
\neg\text{LessThan}(0, 0) & \\
\text{LessThan}(1, 0) & \\
\text{LessThan}(2, 0) & \\
\ldots
\end{align*}
\]

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:

\[ \text{Proc[Clauses]} = \]
\[ \quad \text{If Clauses are unsatisfiable} \]
\[ \quad \text{then return YES} \]
\[ \quad \text{else return NO} \]

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

θ is a most general unifier of literals $l_1$ and $l_2$ iff

1. θ unifies $l_1$ and $l_2$

2. for any other unifier $\theta'$, there is 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. $\{P(a, f(a, g(z)), \ldots P(a, f(a, u, \ldots\}$
      disagreement set, $DS = \{u, g(z)\}$

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

B&L (2005)
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
  \{ $c\theta | c \in S$ and $\theta$ replaces the variables in $c$ by terms from the Herbrand universe $\}\}$

Theorem: $S$ is satisfiable iff Herbrand base is (applies 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 \(\square\) 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 \(N \times N\) chessboard all safely?
    …

• Satisfiability is strongly believed
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

B&L (2005)
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 $\leftarrow []$
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 []
Strategies 3

5. Special treatment for equality
   • instead of using axioms for =, reflexivity, symmetry, transitivity, substitution of equals for equals
   • use new inference rule: paramodulation
   • from \{((t = s)) \cup C_1\} and \{P(\ldots t' \ldots)\} \cup C_2\text{ where } t\theta = t'\theta
   • infer \{P(\ldots s \ldots)\}\theta \cup C_1\theta \cup C_2\theta.
   • collapses many resolution steps into one; see also: theory resolution (later)

6. Sorted logic
   • terms get sorts:
     \[ x:\text{Male} \quad \text{mother:}\{\text{Person} \rightarrow \text{Female}\} \]
   • keep taxonomy of sorts
   • refuse to unify \( P(s) \) 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 $q$ (forward)
  to prove $q$, prove $p$ (backward)
  procedural reading of $\rightarrow$

- 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: $\text{Battleship}(x) \rightarrow \text{Gray}(x)$
  do not want to try to prove something is gray by proving it is a battleship
  
  backward: $\text{Human}(x) \rightarrow \text{Has}(x, \text{spleen})$
  do not want to conclude from someone being human,
  that she has each property

- the basis for the procedural representations

B&L (2005)