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

$\neg, \wedge, \vee, \exists, \forall$

Knowledge Level:

given KB, α, 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 $\models [\{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_k\} \rightarrow \alpha]$

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.

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
  $\bar{p} \Rightarrow \neg p \qquad \overline{\neg p} \Rightarrow p$

- 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 $((p \vee \neg q) \wedge r \wedge s)$
- [] 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 $\chi \vee \Phi$ and $\neg\Psi \vee \zeta$

$$\chi \vee \Phi \qquad\qquad \neg\Psi \vee \zeta$$

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

- Where θ is a unifier for atomic formulae Φ and Ψ
- $\chi \vee \zeta$ 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 $I \models (p \vee \alpha)$ and $I \models (\neg p \vee \beta)$

- Case 1: $I \models p$
  - then $I \models \beta$, so $I \models (\alpha \vee \beta)$.
- Case 2: $I \not\models p$
  - then $I \models \alpha$, so $I \models (\alpha \vee \beta)$.
- Either way, $I \models (\alpha \vee \beta)$.
- So: $\{(p \vee \alpha), (\neg p \vee \beta)\} \models (\alpha \vee \beta)$.

Special case:

- $[p]$ and $[\neg p]$ resolve to $[]$,
- so $\{[p], [\neg p]\} \models$ 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., $\neg p \models (\neg p \vee \neg q)$
- but no derivation

However, . . .

Resolution *is* sound and complete for [] !

- Theorem: $S \vdash []$ iff $S \models []$
  - Result will carry over to quantified clauses (later)

So for any set S of clauses:

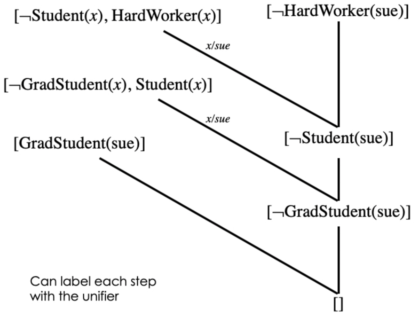
- S is unsatisfiable iff $S \vdash []$.
  - Provides method for determining satisfiability:
    - Search all derivations to see if [] is produced
  - Also provides method for determining all entailments

Example

KB:

$\forall x$ GradStudent(x) $\rightarrow$ Student(x)
$\forall x$ Student(x) $\rightarrow$ HardWorker(x)
GradStudent(sue)

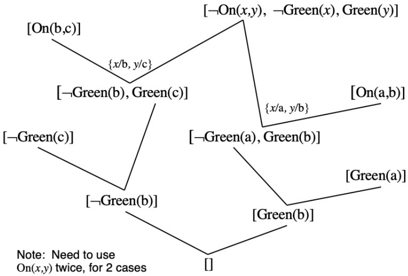
Q: HardWorker(sue)



The 3 block example

KB = {On(a,b), On(b,c), Green(a), ¬Green(c)}
already in CNF

Q = $\exists x \exists y$ [On(x,y) ∧ Green(x) ∧ ¬Green(y)]
Note: ¬Q has no existentials to eliminate;
yields {[¬On(x,y), ¬Green(x), Green(y)]} in CNF



Arithmetic

KB:

Plus(zero,x,x)

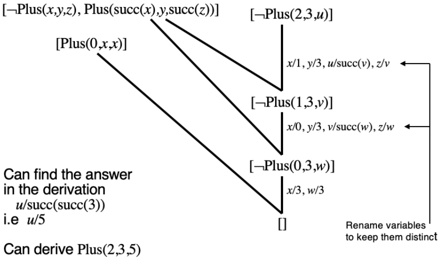
Plus(x,y,z) $\rightarrow$ Plus(succ(x),y,succ(z))

Q: $\exists u$ Plus(2,3,u)

where for readability, we use

0 for zero,

3 for succ(succ(succ(zero))) etc.



B&L (2005)

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

$\exists x \exists y\,[\mathrm{On}(x,y) \wedge \mathrm{Green}(x) \wedge \neg\mathrm{Green}(y)]$

but cannot derive which block is which

Solution: answer-extraction process

replace query $\exists x P(x)$ by $\exists x[P(x) \wedge \neg A(x)]$,

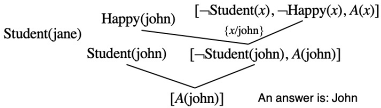
where A is a new predicate symbol called the *answer predicate*

instead of deriving [], derive any clause containing just the answer predicate

can always convert a derivation of []

Example KB: $\{\mathrm{Student(john)}, \mathrm{Student(jane)}, \mathrm{Happy(john)}\}$

Q: $\exists x\,[\mathrm{Student}(x) \wedge \mathrm{Happy}(x)]$

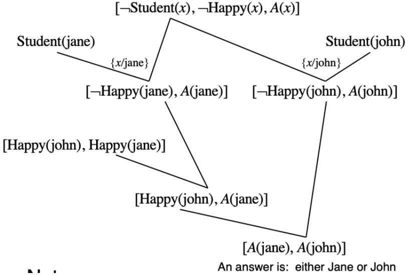


B&L (2005)

Disjunctive answers

Example KB: {Student(john), Student(jane), [Happy(john) $\vee$ Happy(jane)]}

Q: $\exists x$ [Student(x) $\wedge$ Happy(x)]



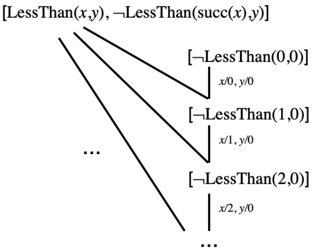
Note:

can have variables in answer

need to watch for Skolem symbols . . .

A Problem

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



Infinte branch of resolvents
  cannot use a simple depth-first procedure to search for []

B&L (2005)

Undecidability

Is there a way to detect when this happens?

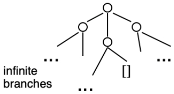
No! FOL is very powerful

- 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:

Proc[Clauses] =
If Clauses are unsatisfiable
then return YES
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:

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

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

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 θ', there is another substitution θ^* such that $\theta' = \theta\theta^*$

   note: composition $\theta\theta^*$ requires applying θ^* to terms in θ

   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. $P(a, f(a, g(z), \ldots \quad P(a, f(a, u, \ldots$ 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

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 θ 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 [] 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?
    $\ldots$
- 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

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 ¬Q
   - always resolve with at least one clause that has an ancestor in ¬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 =, relexitivity, symmetry, transitivity, substitution of equals for equals
   - use new inference rule: paramodulation
   - from $\{(t = s)\} \cup C_1$ and $\{P(\ldots t' \ldots)\} \cup C_2$ 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:Male   mother:[Person $\rightarrow$ 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: Battleship(x) $\rightarrow$ Gray(x)
  do not want to try to prove something is gray by proving it is a battleship
  backward: Human(x) $\rightarrow$ Has(x,spleen)
  do not want to conclude from someone being human,
  that she has each property

- the basis for the procedural representations