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 $\models [(\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_k) \supset \alpha]$

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

$p \Rightarrow \neg p$ $\quad\quad$ $\neg p \Rightarrow p$

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 <u>conjunction</u> of clauses

Clause understood as <u>disjunction</u> of literals

Literals understood normally

So:

$\{[p, \neg q], [r], [s]\}$ is representation of $((p \vee \neg q) \wedge r \wedge s)$

[] is a representation of FALSE

{} is a representation of TRUE

CNF and DNF

Every propositional wff α can be converted into a formula α' in <u>C</u>onjunctive <u>N</u>ormal <u>F</u>orm (CNF) in such a way that $\models \alpha \equiv \alpha'$.

1. eliminate $\supset$ and $\equiv$

   using $(\alpha \supset \beta)$ ß $(\neg\alpha \vee \beta)$ etc.

2. push $\neg$ inward

   using $\neg(\alpha \wedge \beta)$ ß $(\neg\alpha \vee \neg\beta)$ etc.

3. distribute $\vee$ over $\wedge$

   using $((\alpha \wedge \beta) \vee \gamma)$ ß $((\alpha \vee \gamma) \wedge (\beta \vee \gamma))$

4. collect terms

   using $(\alpha \vee \alpha)$ ß α etc.

Result is a conjunction of disjunction of literals.

an analogous procedure produces DNF,
a disjunction of conjunction of literals

We can identify CNF wffs with clausal formulas

$(p \vee \neg q \vee r) \wedge (s \vee \neg r)$ ß $\{[p, \neg q, r], [s, \neg r]\}$

So: given a finite KB and α, to find out if KB $\models \alpha$, it will be sufficient to

1. put (KB $\wedge \neg\alpha$) into CNF, as above

2. determine the satisfiability of clauses

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 <u>resolvent</u> 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 <u>derivation</u> of a clause c from a set *S* of clauses is a sequence $c_1, c_2, ..., c_n$ 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

Rationale

Resolution is a symbol-level rule of inference, but has a connection to knowledge-level logical interpretations

Resolvent is <u>entailed</u> by input clauses.

Suppose $\boldsymbol{I} \models (p \vee \alpha)$ and $\boldsymbol{I} \models (\neg p \vee \beta)$

Case 1: $\boldsymbol{I} \models p$

then $\boldsymbol{I} \models \beta$, so $\boldsymbol{I} \models (\alpha \vee \beta)$.

Case 2: $\boldsymbol{I} \not\models p$

then $\boldsymbol{I} \models \alpha$, so $\boldsymbol{I} \models (\alpha \vee \beta)$.

Either way, $\boldsymbol{I} \models (\alpha \vee \beta)$.

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 <u>is</u> 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

A procedure for entailment

To determine if KB |= α,

- put KB, ¬α into CNF to get *S*, as before
- check if *S* ⊢ [].

If KB = {}, then we are testing the validity of α.

Non-deterministic procedure

1. Check if [] 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_3 not already in *S*.
   If no, then return **SATISFIABLE**
3. Add c_3 to *S* and go to 1.

Note: need only convert KB to CNF once

- can handle multiple queries with same KB
- after addition of new fact α, can simply add new clauses α′ to KB

Example 1

KB:

FirstGrade

FirstGrade ⊃ Child

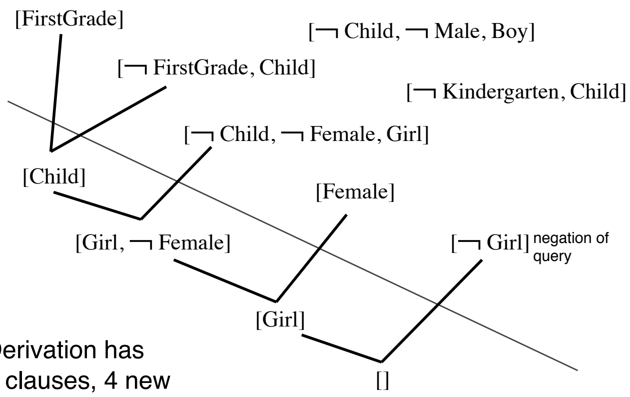
Child ∧ Male ⊃ Boy

Kindergarten ⊃ Child

Child ∧ Female ⊃ Girl

Female

Show that KB |= Girl



Derivation has
9 clauses, 4 new

Example 2

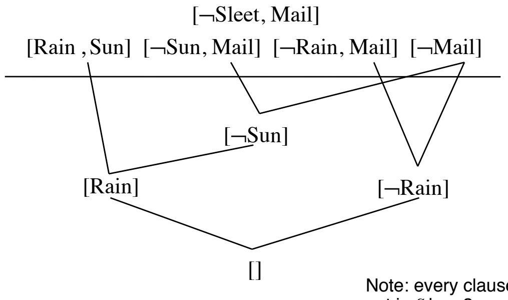
KB:

(Rain $\vee$ Sun)

(Sun $\supset$ Mail)

((Rain $\vee$ Sleet) $\supset$ Mail)

Show KB |= Mail



Note: every clause not in S has 2 parents

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

interpreted as

$$\forall x \forall y \{[R(a,f(b,x)) \supset P(x)] \wedge Q(x,y)\}$$

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

Notation: If l is a literal and θ 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 θ, $l = l'\theta$.

Generalizing CNF

Resolution will generalize to handling variables

But how to convert wffs to CNF?

Need three additional steps: Ignore = for now

1. eliminate $\supset$ and $\equiv$
2. push $\neg$ inward

   using also $\neg\forall x.\alpha$ ß $\exists x.\neg\alpha$ etc.

3. standardize variables: each quantifier gets its own variable

   e.g. $\exists x[P(x)] \wedge Q(x)$ ß $\exists z[P(z)] \wedge Q(x)$ where z is a new variable

4. eliminate all existentials

   (discussed later)

5. move universals to the front

   using $\forall x[\alpha] \wedge \beta$ ß $\forall x[\alpha \wedge \beta]$ where β does not use x

6. distribute $\vee$ over $\wedge$
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 θ such that $l_1\theta = l_2\theta$, can infer $(C_1 \cup C_2)\theta$. example below

We say that l_1 unifies with l_2 and that θ is a unifier of the two literals

Resolution derivation: as before still ignoring =

Theorem: $S \vdash []$ iff $S \models []$

iff S is unsatisfiable

Example 3

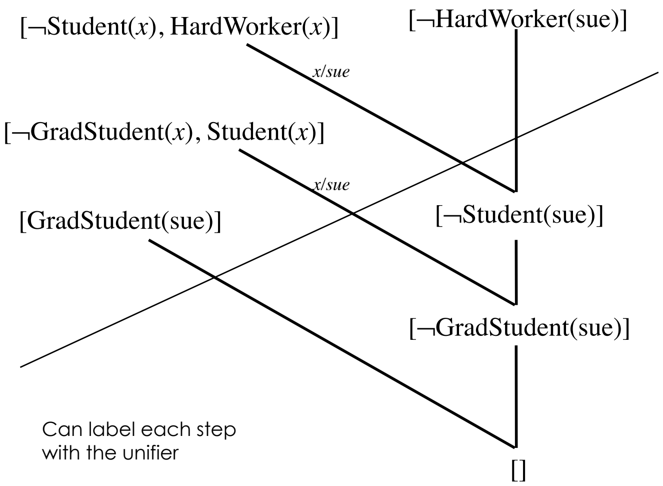
KB:

$\forall x\ \mathrm{GradStudent}(x) \supset \mathrm{Student}(x)$

$\forall x\ \mathrm{Student}(x) \supset \mathrm{HardWorker}(x)$

GradStudent(sue)

Q: HardWorker(sue)



The 3 block example

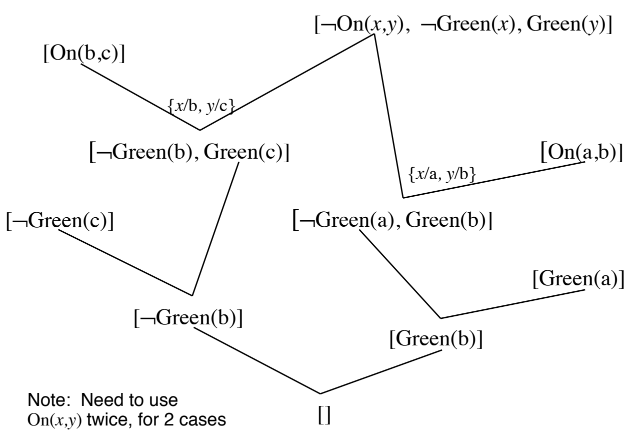
KB = {On(a,b), On(b,c), Green(a), ¬Green(c)}

already in CNF

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

Note: ¬Q has no existentials to eliminate

yields $\{[\neg\,\mathrm{On}(x,y), \neg\,\mathrm{Green}(x), \mathrm{Green}(y)]\}$ in CNF



Arithmetic

KB:

$\text{Plus}(\text{zero},x,x)$
$\text{Plus}(x,y,z) \supset \text{Plus}(\text{succ}(x),y,\text{succ}(z))$

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

where for readability, we use
0 for zero,
3 for succ(succ(succ(zero))) etc.

$[\neg\text{Plus}(x,y,z), \text{Plus}(\text{succ}(x),y,\text{succ}(z))]$

$[\text{Plus}(0,x,x)]$

$[\neg\text{Plus}(2,3,u)]$

$x/1,\ y/3,\ u/\text{succ}(v),\ z/v$

$[\neg\text{Plus}(1,3,v)]$

$x/0,\ y/3,\ v/\text{succ}(w),\ z/w$

$[\neg\text{Plus}(0,3,w)]$

$x/3,\ w/3$

$[]$

Rename variables to keep them distinct

Can find the answer in the derivation
$u/\text{succ}(\text{succ}(3))$
i.e $u/5$

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

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

but cannot derive which block is which

Solution: answer-extraction process

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

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

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

$\Downarrow$

$[\neg\text{Student}(x), \neg\text{Happy}(x), A(x)]$

Happy(john)

$\{x/\text{john}\}$

Student(jane)

Student(john)

$[\neg\text{Student}(\text{john}), A(\text{john})]$

$[A(\text{john})]$

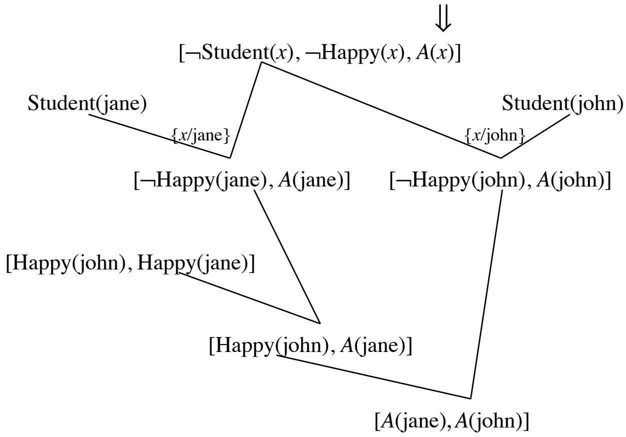
An answer is: John

Disjunctive answers

Example KB:

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

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



An answer is: either Jane or John

Note:

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

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 <u>Skolem</u> 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:

$$\forall x_1(\ldots\forall x_2(\ldots\forall x_n(\ldots\exists y[\ldots\ y\ \ldots]\ \ldots)\ldots)\ldots)$$

is replaced by

$$\forall x_1(\ldots\forall x_2(\ldots\forall x_n(\ \ldots\ [\ldots f(x_1,x_2,\ldots,x_n)\ \ldots]\ \ldots)\ldots)\ldots)$$

where f is a new function symbol that appears nowhere else

Skolemization does <u>not</u> preserve equivalence

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

But it does preserve satisfiability

α is satisfiable iff α' is satisfiable

where α' 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

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

$\neg \forall y \exists x R(x,y)$ ß $\exists y \forall x \neg R(x,y)$ ß $\forall x \neg R(x,b)$

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

Show that $\forall y \exists x R(x,y) \not\models \exists x \forall y R(x,y)$

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

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

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

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

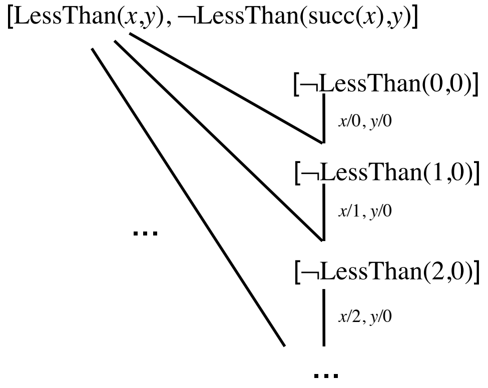
first argument depends on second one here

A problem

KB: $\text{LessThan}(\text{succ}(x),y) \supset \text{LessThan}(x,y)$

Q: $\text{LessThan}(\text{zero},\text{zero})$

Should fail since KB $\not\models$ Q



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:

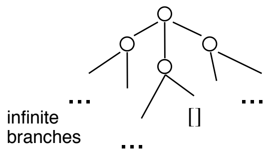
Proc[*Clauses*] =

If *Clauses* are unsatisfiable
then return YES
else return NO

Also true for Horn clauses (later)

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 <u>most general unifier</u> of literals l_1 and l_2 iff

1. θ unifies l_1 and l_2
2. for any other unifier θ', there is a 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 $P(a,f(a,g(z),\ldots$ $P(a,f(a,u,\ldots$

   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 <u>linear</u> algorithm

Herbrand Theorem

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

Given a set of clauses *S*

- the <u>Herbrand universe</u> 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 <u>Herbrand base</u> of *S* is

  $\{\, c\theta \mid c \in S$ 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 <u>propositional</u>, 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 $\{c_1, c_2, ..., c_n\}$ 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×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 ß []

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, ... 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: <u>paramodulation</u>

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)

[father(john)=bill] [Married(father(x),mother(x))]

[Married(bill,mother(john))]

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 $[\sim p, q]$, can interpret as either

from p, infer q (forward)

to prove q, prove p (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,spleen)

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

the basis for the procedural representations