

Knowledge Representation & Reasoning

Shyamanta M Hazarika

CSE, SoE

Tezpur University

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

So replace $\forall x_1 (\dots \forall x_2 (\dots \forall x_n (\dots \exists y [\dots y \dots] \dots) \dots) \dots)$

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

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

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

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

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

then $\{ [R(a,y)], [\neg R(x,b)] \} \rightarrow []$ 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

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

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

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

A problem

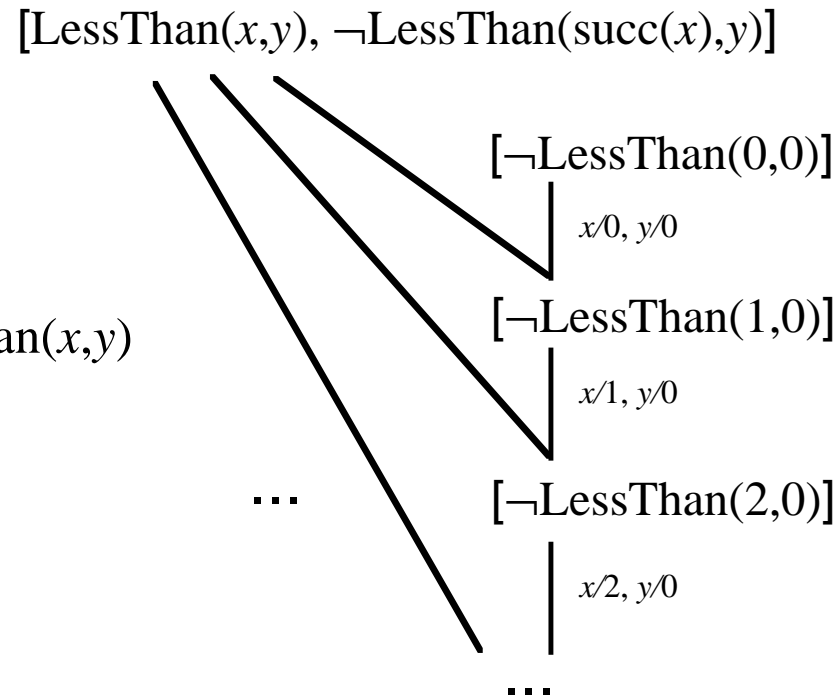
KB:

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

Query:

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

Should fail since $\text{KB} \not\models Q$



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:

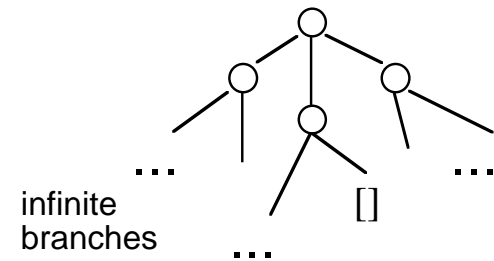
```
Proc[Clauses] =  
  If Clauses are unsatisfiable  
  then return YES  
  else return NO
```

However: Resolution is complete

some branch will contain [], for unsatisfiable 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 thing that can be done:

reduce redundancy in search, by keeping search as general as possible

Example

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

unified by

$$\theta_1 = \{x/b, y/g(b), z/a, w/b\} \text{ gives } P(g(b), f(b), a)$$

and by

$$\theta_2 = \{x/f(z), y/g(f(z)), z/a, w/f(z)\} \text{ gives } P(g(f(z)), f(f(z)), a).$$

Might not be able to derive the empty clause from clauses having overly specific substitutions

wastes time in search!

Most general unifiers

θ is a most general unifier (MGU) of literals ρ_1 and ρ_2 iff

1. θ unifies ρ_1 and ρ_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 most general unifiers only without loss of completeness (with certain caveats)

Computing MGUs

Computing an MGU, given a set of literals $\{\rho_i\}$

usually only have two literals

1. Start with $\theta := \{\}$.
2. If all the $\rho_i\theta$ are identical, then done;
otherwise, get disagreement set, DS
e.g $P(a, f(a, g(z), \dots)) \quad P(a, f(a, u, \dots)$
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 in S (at least one)

e.g., if S uses (unary) f , and c, d , $U = \{c, d, f(c), f(d), f(f(c)), f(f(d)), f(f(f(c))), \dots\}$

- the Herbrand base of S is the set of all $c\theta$ such that $c \in S$ 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?

Shown by Haken in 1985 that there are unsatisfiable clauses $\{c_1, c_2, \dots, c_n\}$ 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 a set of clauses is satisfiable was shown by Cook in 1972 to be NP-complete

No easier than an extremely large variety of computational tasks

Roughly: 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 $\leq k$ miles?
- » can N queens be put on an $N \times N$ chessboard all safely? and many, many more....

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

SAT solvers

In the propositional case, procedures have been proposed for determining the satisfiability of a set of clauses that appear to work much better in practice than Resolution.

The most popular is called DP (or DPLL) based on ideas by Davis, Putnam, Loveland and Logemann.

These procedures are called SAT solvers as they are mostly used to find a satisfying interpretation for clauses that are satisfiable.

related to constraint satisfaction programs (CSP)

Typically they have the property that if they *fail* to find a satisfying interpretation, a Resolution derivation of [] can be reconstructed from a trace of their execution.

so worst-case exponential behaviour, via Haken's theorem!

One interesting counter-example to this is the procedure GSAT, which has different limitations.

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 e.g. procedural representations
- less expressive languages e.g. Horn clauses

In some applications, it is reasonable to wait

e.g. mathematical theorem proving, where we care about specific formulas

Best to hope for in general: reduce redundancy

main example: MGU, as before

but many other strategies (as we will see)

ATP: automated theorem proving

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