Resolution (continuation)

Answer extraction

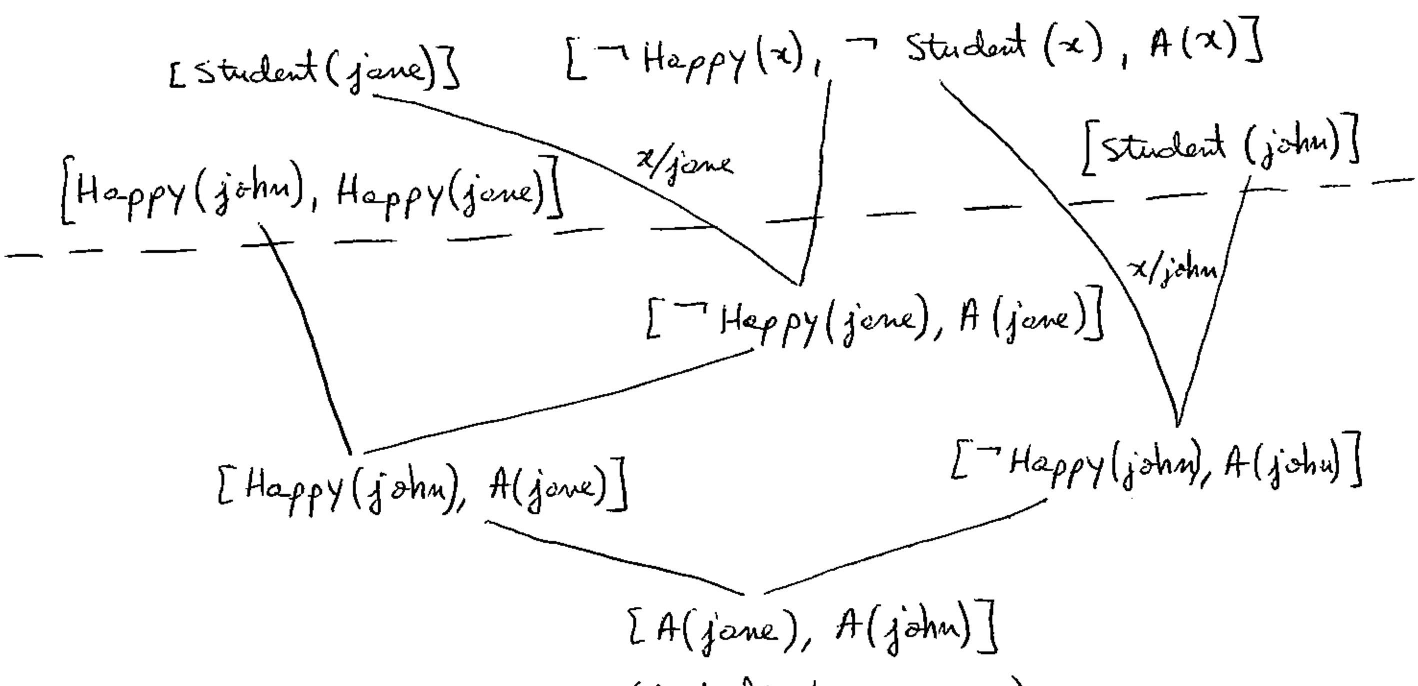
It is often possible to get answers to questions by looking at the bindings of variables in a derivation of an existential (see Example 5 in the previous course). But more complicated situations may appear in FOL. In Example 4 (prev. course), we know that there is a block that satisfies a condition, but we don't know which block. That is to say that $KB \models \exists x. P(x)$ without entailing $P(t)$ for a specific t.

Idea: replace a question such as $\exists x. P(x)$ by $\exists x. P(x) \land \neg A(x)$, where A is a new predicate symbol that occurs nowhere else. A is called the answer predicate. Since A does not appear anywhere else, it will not be possible to derive the empty clause. Therefore, the derivation ends when we produce a clause containing only the answer predicate.

Example 1

$$KB \begin{cases} Student(john) \\ Student(jane) \\ Happy(john) \lor Happy(jane) \end{cases}$$

Question: $\exists x. Student(x) \land Happy(x)$



(indefinite answer)

<u>Obs</u>. We can get answers containing variables

For example,

$$KB \left[\begin{array}{l} \forall z.\ Child(f(g(a)), z) \\ \forall x \forall y.\ Play(f(x), g(y)) \end{array} \right.$$

Question: $\exists x \exists y.\ Child(x,y) \wedge Play(x,y)$

The negation of the question is $[\neg Child(x,y), \neg Play(x,y), A(x,y)]$ and we get a derivation with the final clause $[A(f(g(a)), g(y))]$ interpreted as "answer is any instance of the terms $f(g(a)), g(y)$".

<u>Skolemization</u>

It is the process of removing existential quantifiers by elimination.

<u>idea</u>: we introduce unique names for each variable quantified with an existential.

$$\left. \begin{array}{l} \exists x \forall y \exists z.\ P(x,y,z) \\ \text{we name } x \quad a \\ \qquad\qquad\quad z \quad f(y) \end{array} \right\} \begin{array}{l} \text{we will use instead} \\ \forall y.\ P(a, y, f(y)) \end{array}$$

a, f are called Skolem symbols (they do not appear anywhere else).

In general, Skolemization replace each existential variable by a new function symbol with as many arguments as there are universal variables dominating the existential.

$$\alpha \quad \forall x_1 (\ldots \forall x_2 (\ldots \forall x_3 (\ldots \exists y [\ldots y \ldots] \ldots) \ldots) \ldots)$$

$$\Big\downarrow \text{ skolemization}$$

$$\alpha' \quad \forall x_1 (\ldots \forall x_2 (\ldots \forall x_3 (\ldots [\ldots f(x_1, x_2, x_3) \ldots] \ldots) \ldots) \ldots)$$

where f appears nowhere else.

<u>Obs</u>. $\not\models (\alpha \equiv \alpha')$

For example, $\exists x.\, Kill(x, victim)$ strictly speaking is not logically equivalent to $Kill(murderer, victim)$.

<u>Att</u>: Do not confuse these substitutions (i.e. Skolem constants) with the interpretations used to define the semantics of quantifiers. The substitution replaces a variable with a term (it's syntax) to produce new sentences, whereas an interpretation maps a variable to an object in the domain.

<u>Obs</u>: It can be proven that α and α' are inferentially equivalent:

$$KB \cup \{\alpha\} \text{ is satisfiable iff } KB \cup \{\alpha'\} \text{ is satisfiable}$$

<u>Att</u>: For logical correctedness, it is important to have the dependence of variables right.

For example, $\exists x \forall y\, R(x,y) \models \forall y \exists x\, R(x,y)$ but the converse does not hold.

$$\{\exists x \forall y\, R(x,y),\ \neg \forall y \exists x\, R(x,y)\}$$
$$\downarrow \text{CNF}$$
$$\{[R(a,y)],\ [\neg R(x,b)]\} \qquad a, b \text{ Skolem constants}$$
$$\downarrow x/a,\ y/b$$
$$[\ ]$$

The converse $\{\forall y \exists x\, R(x,y),\ \neg \exists x \forall y\, R(x,y)\}$
$$\downarrow \text{CNF}$$
$$\{[R(f(y),y)],\ [\neg R(x,g(x))]\} \qquad f, g \text{ Skolem functions}$$

do not unify

Example 2 (taken from https://www.cs.utexas.edu/users/novak/reso.html)

KB:
- All hounds howl at night.
- Anyone who has any cats will not have any mice.
- Light sleepers do not have anything that howls at night.
- John has either a cat or a hound.

Question: If John is a light sleeper, then John does not have any mice.

KB:

$$\forall x\ (Hound(x) \supset Howl(x))$$

$$\forall x \forall y \forall z\ ((Have(x,y) \wedge Cat(y)) \supset \neg (Have(x,z) \wedge Mouse(z)))$$

$$\forall x \forall y\ (Ls(x) \supset \neg (Have(x,y) \wedge Howl(y)))$$

$$\exists x\ (Have(john, x) \wedge (Cat(x) \vee Hound(x)))$$

Q: $\forall x\ (Ls(john) \supset \neg (Have(john, x) \wedge Mouse(x)))$

$$\Big\downarrow \text{CNF}$$

$[\neg Hound(x), Howl(x)]$

$[\neg Have(x,y), \neg Cat(y), \neg Have(x,z), \neg Mouse(z)]$

$[\neg Ls(x), \neg Have(x,y), \neg Howl(y)]$

$[Have(john, a)]$ a - Skolem constant

$[Cat(a), Hound(a)]$

$[Ls(john)]$

$[Have(john, b)]$ b - Skolem constant

$[Mouse(b)]$

... apply Resolution

$[\ ]$

Equality

if we treated = as a predicate, we would miss, for example, that $\{a=b, b=c, a\neq c\}$ is unsatisfiable. For that reason, it is necessary to add the clausal versions of the axioms of equality:

- reflexivity $\quad \forall x.\ x=x$
- symmetry $\quad \forall x \forall y.\ x=y \supset y=x$
- transitivity $\quad \forall x \forall y \forall z.\ x=y \wedge y=z \supset x=z$
- substitution for functions

$$\forall x_1 \forall y_1 \ldots \forall x_n \forall y_n.\ x_1=y_1 \wedge \ldots \wedge x_n=y_n \supset$$
$$f(x_1,\ldots,x_n)=f(y_1,\ldots,y_n)$$ for every function symbol f of arity n

- substitution for predicates

$$\forall x_1 \forall y_1 \ldots \forall x_n \forall y_n.\ x_1=y_1 \wedge \ldots \wedge x_n=y_n \supset$$
$$P(x_1,\ldots,x_n)\equiv P(y_1,\ldots,y_n)$$ for every predicate symbol P of arity n

Now = can be treated as a binary predicate

Example 3

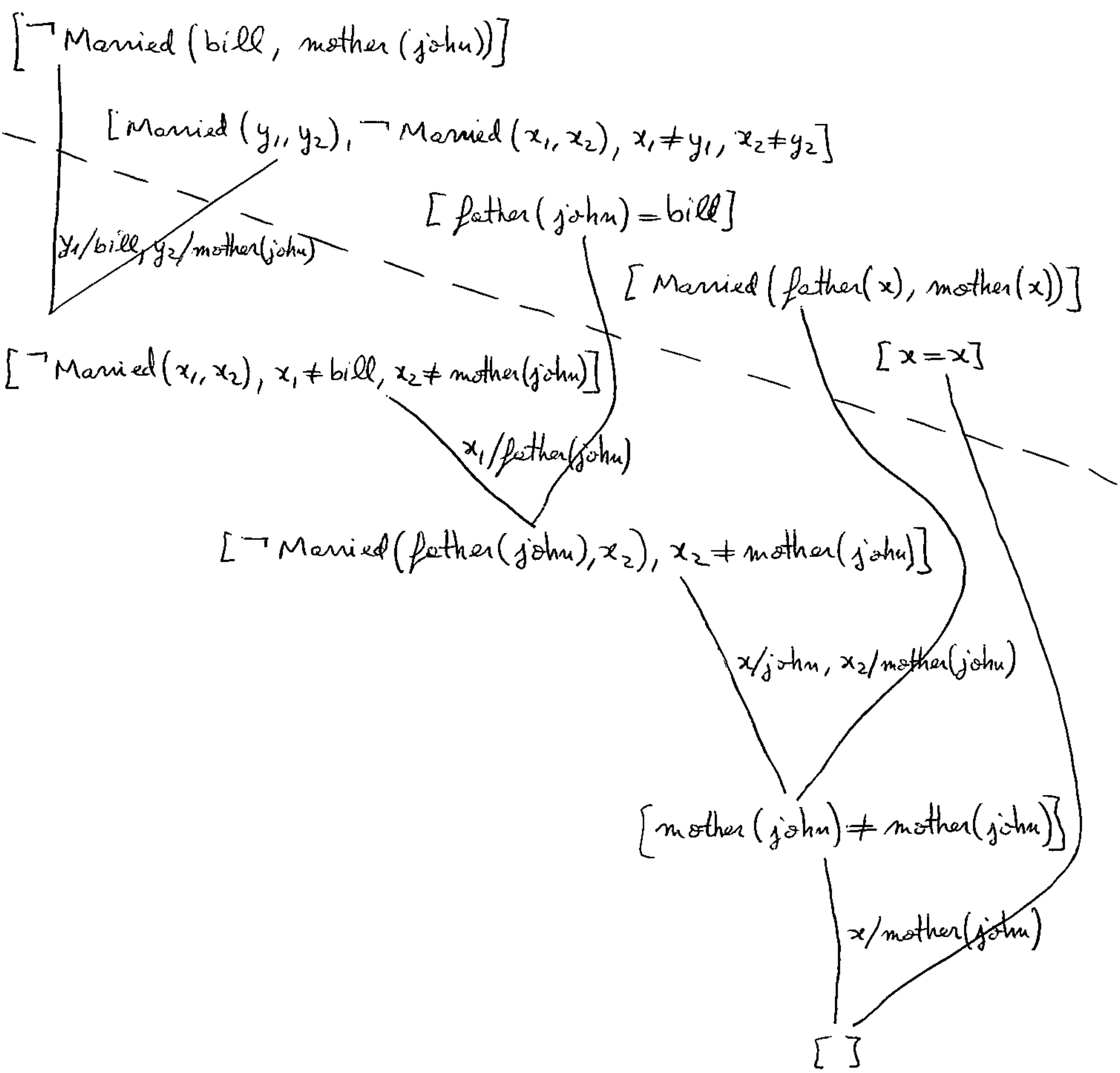
KB $\left[\begin{array}{l}\forall x.\ Married(father(x), mother(x)) \\ father(john) = bill\end{array}\right.$

Question: $Married(bill, mother(john))$

$$\forall x_1 \forall y_1 \forall x_2 \forall y_2.\ x_1=y_1 \wedge x_2=y_2 \supset Married(x_1,x_2)\equiv Married(y_1,y_2)$$

$\Big\downarrow$ CNF (replace $\supset$ and $\equiv$; distribute $\wedge$ over $\vee$; and collect terms)

$$[Married(y_1,y_2), \neg Married(x_1,x_2), x_1\neq y_1, x_2\neq y_2]$$



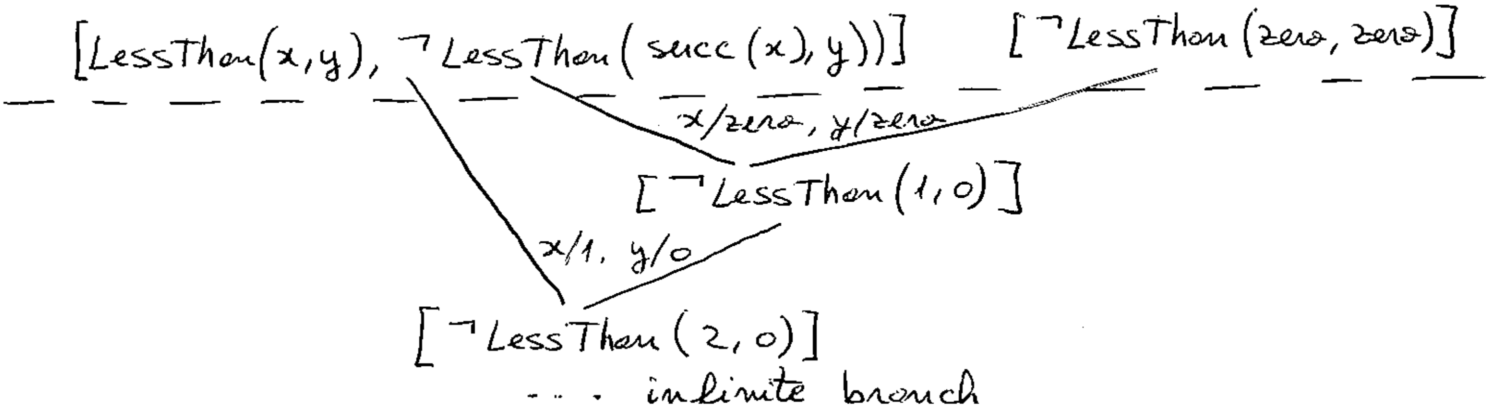
<u>Dealing with computational intractability</u>

Resolution does not provide a general effective solution for automated reasoning.

<u>The FOL Case</u>

KB: $\forall x \forall y.\ LessThan(succ(x), y) \supset LessThan(x, y)$

Question: $LessThan(zero, zero)$.



if we apply a depth-first procedure to search for the empty clause, we can go on an infinite branch.
We cannot detect whether a branch will continue indefinitely. But we know that $S \models [\,]$ iff $S \vdash [\,]$, meaning that if a set of sentences is unsatisfiable, then there is a branch in the derivation containing the empty clause. So, a breadth-first search would guarantee to find it (the unsatisfiable case). But when the clauses are satisfiable, the search may or may not end.

The Herbrand Theorem

In the propositional case, Resolution procedure always terminates (in the initial set of clauses there is a finite number of literals).
In some cases, Resolution in FOL reduces to the propositional case.

Def. Given S a set of clauses, the Herbrand universe of S, written H_S, is the set of all ground terms (i.e. terms with no variables) formed using just the constants and the function symbols in S (if S has no constants or function symbols, we use just a constant a).

For example, if $S = \{[\neg P(x, f(x,a)), \neg Q(x,a), R(x,b)]\}$

then $H_S = \{a, b, f(a,a), f(a,b), f(b,a), f(b,b), f(a, f(a,a)), \ldots\}$

Def. The Herbrand base of S, written $H_S(S)$, is the set of all ground clauses $c\theta$, where $c \in S$ and θ assigns the variables in c to terms in the Herbrand universe.

For the same S as before, we have:

$$H_S(S) = \{ [\neg P(a, f(a,a)), \neg Q(a,a), R(a,b)],$$
$$[\neg P(b, f(b,a)), \neg Q(b,a), R(b,b)],$$
$$[\neg P(f(a,a), f(f(a,a),a)), \neg Q(f(a,a),a), R(f(a,a),b)],$$
$$[\neg P(f(b,a), f(f(b,a),a)), \neg Q(f(b,a),a), R(f(b,a),b)], \ldots \}$$

<u>Herbrand's Theorem</u> - A set of clauses is satisfiable iff its Herbrand base is satisfiable.

This is important because the Herbrand base is a set of clauses without variables, so it is essentially propositional. The difficulty is that typically the Herbrand base is an infinite set of propositional clauses (but finite when the Herbrand universe is finite - no function symbols and a finite number of constants in S).
Sometimes, the Herbrand univers can be kept finite by looking at the type of the arguments and values of functions, and including terms like $f(t)$ only if the type of t is appropriate for f (e.g. we may exclude terms like birthday(birthday(john)) from the Herbrand universe).

<u>The propositional case</u>

How long may it take for the Resolution procedure on a finite set of propositional clauses to terminate?

In 1985, Armin Haken proved that there exist unsatisfiable propositional clauses $C_1, \ldots, C_n$ so that the shortest derivation of the empty clause has the length of order 2^n.
Resolution takes an exponential time, no matter how well we choose the derivations.

Is there a better way to determine whether a set of propositional clauses is satisfiable? - one of the most difficult questions in computer science.

In 1972, Stephen Cook proved that the satisfiability problem was NP-complete. Any search problem (e.g. scheduling, routing) where we are searching for an element that satisfies a certain property, and where we can test in polynomial time whether a candidate satisfies the property, can be converted into a propositional satisfiability problem. Thus, a polynomial time algorithm for satisfiability would imply a polynomial time for all these search problems.

We may need to consider alternative options for Resolution:

- procedural representations - give more control over the reasoning process to the user.
- using representation languages that are less expressive than FOL - e.g. description languages

The research in KRR approaches both directions.

In some applications it may be worth waiting (for a long time) for answers.

There is an area of AI called automated theorem-proving, that uses Resolution (among other procedures) for this purpose (e.g. to determine whether or not Goldbach's Conjecture follows from the axioms of number theory).

<u>SAT Solvers</u>

They are procedures that determine the satisfiability of a set of clauses more efficiently than the Resolution. They search for an interpretation that would prove the clauses to be satisfiable. They are often applied to clauses that are known to be satisfiable, but the satisfying interpretation is not known.

When C is a set of clauses and m is a literal, $C \bullet m$ is defined as following:

$$C \bullet m = \{c \mid c \in C,\ m \notin c,\ \overline{m} \notin c\} \cup \{(c - \overline{m}) \mid c \in C,\ m \notin c,\ \overline{m} \in c\}$$

For example, if $C = \{[p, q], [\overline{p}, a, b], [\overline{p}, c], [d, e]\}$

then $C \bullet p = \{[a, b], [c], [d, e]\}$

$C \bullet \overline{p} = \{[q], [d, e]\}$

$(C \bullet \overline{p}) \bullet \overline{q} = \{[\,], [d, e]\}$

$(C \bullet \overline{p}) \bullet q = \{[d, e]\}$

$((C \bullet \overline{p}) \bullet q) \bullet d = \{\}$

Given an interpretation I,

- if p is true then C is satisfiable iff $C \bullet p$ is satisfiable
- if p is false then C is satisfiable iff $C \bullet \overline{p}$ is satisfiable.

<u>Procedure DP (Davis - Putnam)</u>

input: a set of clauses C

output: are the clauses satisfiable: YES or NO

```
procedure DP(C)
    if (C is empty) then return YES
    if (C contains [ ]) then return NO
    let p be some atom in C
    if (DP(C • p) = YES) then return YES
                         else return DP(C • p̄)
```

Strategies for choosing an atom p:

- p appears in the most clauses in C;
- p appears in the fewest clauses in C;
- p is the most balanced atom in C (i.e. the number of positive occurences in C is closest to the number of negative occurences);
- p is the least balanced atom in C;
- p appears in the shortest clause in C.

For the propositional case, the DP procedure is the fastest one in practice, among all the known SAT solvers. Thus, problems with tens of millions of variables can be approached. SAT solvers revolutionized fields such as hardware verification or security protocols verification.

<u>Most general unifiers</u>

The most efficient way to avoid unnecessary search in a first-order derivation is to keep the search as general as possible.

For example, consider the clause c_1 containing the literal $P(g(x), f(x), z)$ and the clause c_2 with $\neg P(y, f(w), a)$.

For unification, we may have the substitution

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

or

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

or

...

We can try to derive the empty clause using θ_1; if it doesn't work, we try with θ_2 and so on.

θ_1 and θ_2 are more specific than they should be (it is not necessary to give a value for x).

The substitution $\theta_3 = \{y/g(x), z/a, w/x\}$ unifies C_1 and C_2 without making unnecessary arbitrary choices that might exclude a path to the empty clause.
θ_3 is a most general unifier (MGU). It may not be unique — for example $\theta_4 = \{y/g(w), z/a, x/w\}$ is also an MGU.

<u>Def</u>. A most general unifier θ of literals ρ_1 and ρ_2 is a unifier that has the property that for any other unifier θ', there is a substitution θ^* such that $\theta' = \theta \cdot \theta^*$. By $\theta \cdot \theta^*$ we mean that we first apply θ and then apply θ^* to the result.

For example, from θ_3 we can get to θ_1 by further applying x/b; to θ_2 by applying $x/f(z)$; and to θ_4 by applying x/w.
By limiting Resolution to MGUs, the completeness is maintained and the number of resolvents is dramatically reduced.

<u>The procedure for computing an MGU</u>

input: literals ρ_1 and ρ_2
output: a substitution θ

1. $\theta = \{\}$
2. if $(\rho_1\theta = \rho_2\theta)$ then exit
3. determine the disagreement set DS, which is the pair of terms at the first place (from left to right) where the two literals disagree. For example,

   if $\begin{matrix} \rho_1\theta = P(a, f(a, \underline{g(z)}, \ldots) \\ \rho_2\theta = P(a, f(a, \underline{u}, \ldots) \end{matrix}$ then $DS = \{u, g(z)\}$
4. find a variable $v \in DS$ and a term $t \in DS$ not containing v; if none, fail.
5. otherwise, set $\theta = \theta \cdot \{v/t\}$ and go to 2.

The procedure is very efficient in practice.
All Resolution-based systems use MGUs.

Other optimizations to Resolution to improve the search

Clause elimination

There are types of clauses that do not participate in the (shortest) derivation to the empty clause:

- pure clauses – contain some literal ρ such that $\bar{\rho}$ does not appear anywhere else.
- tautologies – contain both ρ and $\bar{\rho}$ and they can be bypassed in any derivation.
- subsumed clauses – clauses for which there already exists another clause with a subset of the literals (i.e. clauses more specific than a clause in KB).
  For example, if $[P(x)] \in KB$ then we do not add $[P(a)]$ or $[P(a), Q(b)]$.
  if we have $[p, r]$, we don't need $[p, q, r]$.

Ordering strategies

- choose a predefined order to perform Resolution to maximize the chance of deriving the empty clause.
- the best strategy up-to-date is "unit preference", that is, to use unit clauses first.
  a unit clause + a clause with k literals $\Rightarrow$ a clause of length $k-1$ ...

Special treatment of equality

The explicit use of the axioms of equality can generate many resolvents. A way to avoid that is by introducing a second rule of inference in addition to Resolution, called Paramodulation.

We are given two clauses:

$C_1 \cup \{t = s\}$ where t and s are terms

$C_2 \cup \{\rho[t']\}$ containing some term t'.

if necessary, we rename the variables in the two clauses to be distinct.
We assume that there is a substitution θ such that $t\theta = t'\theta$.

Then we can infer the clause $(C_1 \cup C_2 \cup \rho[s])\theta$, which eliminates $=$, replaces t' by s and perform substitution θ.

In Example 3, we have

$$[\underbrace{father(john)}_{t} = \underbrace{bill}_{s}] \qquad [\underbrace{Married}_{\rho}(\underbrace{father(x)}_{t'}, mother(x))]$$

$$C_1 = \{\} \qquad C_2 = \{\}$$

$$\theta = \{x/john\}$$

We can derive $[Married(bill, mother(john))]$ in a single Paramodulation step.

Directional connectives

A clause like $[\neg p, q]$ representing $p \supset q$ can be used in two directions in derivation:

- forward - if we derive a clause containing p, then we derive the clause containing q
- backward - if we derive a clause containing $\neg q$, then we derive the clause containing $\neg p$.

We can mark clauses to be used in one or the other direction only (with care not to lose completeness).

For example, if we have in KB

$$\forall x.\ Battleship(x) \supset Gray(x)$$

it may be used only in the forward direction (it might not be such a good idea to prove that something is not a battleship if it is not gray.