backward chaining

- goal driven, 'wishful thinking'
  - theorems are **found** by backward chaining but **presented** via forward chaining

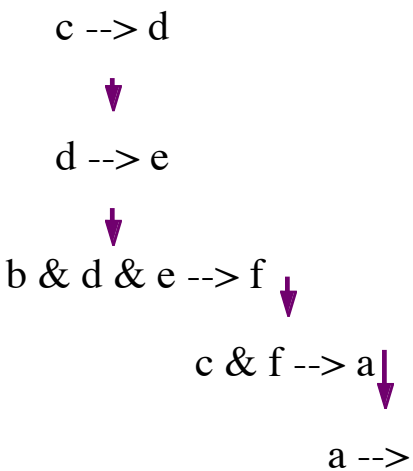
prove **h**, given **b, c**. rules:

1) b & d & e --> f; 2) d & g --> a; 3) c & f --> a; 4) c --> d; 5) d --> e; 6) a --> h

6) prove a; 2) prove d, g; 4) proves d; g impossible, so backup!

3) prove c, f; c given; 1) prove b, d, e; b given; 4) proves d; 5) proves e;

now we have f; 3) proves a, 6) proves h.



resolution

- just one inference rule!

e.g. $\neg A$

$A \leftarrow B$

resolvent: $\neg B$

¬(dark & winter & cold)
winter if january
resolvent: ¬(dark & january & cold)

e.g. $\neg(A_1, \ldots, A_n)$

$A_k \leftarrow B_1, \ldots, B_m \quad (1 \leq k \leq n)$

resolvent: $\neg(A_1, \ldots, A_{k-1}, B_1, \ldots, B_m, A_{k+1}, \ldots A_n)$

e.g.

1) receives(you, power) ← gives(logic, power, you)
2) gives(logic, power, you)
? receives(you, power)
3) **query: ¬ receives(you, power)**
1), 3) imply 4) ¬ gives(logic, power, you)
2), 4) imply □; answer: yes

resolution, cont.

convert fmlas to **clausal form (CNF)**

$\alpha = \alpha_1 \wedge \alpha_2 \ldots \wedge \alpha_n$, where $\alpha_i = \beta_{i,1} \vee \ldots \vee \beta_{i,k}$ and each $\beta_{i,j}$ is a **literal.**

Conversion (in propositional logic) uses

$(\alpha \rightarrow \beta) \leftrightarrow (\neg\alpha \vee \beta)$

$(\alpha \leftrightarrow \beta) \leftrightarrow (\alpha \wedge \beta) \vee (\neg\alpha \wedge \neg\beta)$

$\neg(\alpha \vee \beta) \leftrightarrow (\neg\alpha \wedge \neg\beta)$

$\neg(\alpha \wedge \beta) \leftrightarrow (\neg\alpha \vee \neg\beta)$

$\neg\neg\alpha \leftrightarrow \alpha$

$((\alpha \wedge \beta) \vee \gamma) \leftrightarrow ((\alpha \vee \gamma) \wedge (\beta \vee \gamma))$

e.g. convert $((p \rightarrow q) \rightarrow (p \wedge q))$:

$\neg(p \rightarrow q) \vee (p \wedge q)$, $\neg(\neg p \vee q) \vee (p \wedge q)$,

$(p \wedge \neg q) \vee (p \wedge q)$, $(p \vee p) \wedge (p \vee q) \wedge (\neg q \vee p) \wedge (\neg q \vee q)$,

$(p \vee q) \wedge (\neg q \vee p)$, i.e. $\{p,q\}, \{\neg q, p\}$.

resolution, cont.

- If clause C_1 contains p and C_2 contains ¬p, then the **resolvent of C_1 and C_2 on p** is a clause containing all <u>other</u> elements of C_1 and C_2.
- Principle of (propositional) resolution:

$\therefore$ $((\mathbf{p} \vee \alpha) \wedge (\mathbf{\neg p} \vee \beta)) \rightarrow (\alpha \vee \beta)$

¬p → α, p → β, and since p ∨ ¬p:
α ∨ β, i.e ¬ α → β.
…. transitivity of implication

- algorithm: to derive α from premisses S:

convert S and ¬α to clausal form (and aim for a contradiction)
repeat
    <u>pick</u> p and C_1, C_2 that can be resolved on p
    simplify resolvent by eliminating duplicates
    remove resolvent if it has both a q and ¬q
    add resolvent to original set if not there
if the empty clause results, S implies α.

conversion to clausal form

- e.g. $\{c \rightarrow s, \neg g \rightarrow d, \neg g \vee c\}$ implies $\neg s \rightarrow d$???

$\neg c \vee s, g \vee d, \neg g \vee c$, neg of conclusion: $\neg s \wedge \neg d$

$C_1 = \{\neg c, s\}, C_2 = \{g, d\}, C_3 = \{\neg g, c\}, C_4 = \{\neg s\}, C_5 = \{\neg d\},$

$(((C_2 * C_5) * C_3) * C_1) * C_4) = \square$ = empty clause (successful derivation)

- e.g. $\{\neg p \rightarrow r, p \rightarrow s, r \rightarrow q, s \rightarrow \neg t, t\}$ implies q??

$C_1 = \{p, r\}, C_2 = \{\neg p, s\}, C_3 = \{\neg r, q\}, C_4 = \{\neg s, \neg t\}, C_5 = \{t\}, C_6 = \{\neg q\}.$

$\ldots (((C_3 * C_6) * C_1) * C_2) * C_4) * C_5 =$ empty clause $\square$

- e.g. prove $p \vee \neg p$: negate: $\neg p \wedge p$; $\{\neg p\}, \{p\}$ resolves to empty clause.

Conversion to clausal form is a bit trickier in fol:

$$\neg \forall \upsilon \, \Phi \Leftrightarrow \exists \upsilon \, \neg \Phi, \quad \neg \exists \upsilon \, \Phi \Leftrightarrow \forall \upsilon \, \neg \Phi$$

standardize variables apart: $\forall x Fx \wedge \exists x Gx$ becomes $\forall x Fx \wedge \exists y Gy$

converting fol to clausal form

- $\neg\forall\upsilon\ \Phi \Leftrightarrow \exists\upsilon\ \neg\Phi,\ \neg\exists\upsilon\ \ \Phi \Leftrightarrow \forall\upsilon\ \neg\Phi$
- remove $\exists$:

  if $\exists$ is not in scope of universal quantifier, drop it and replace quantified variable by **new** constant,

  else … by a term formed from **new function symbol** applied to variables associated with enclosing universal quantifiers (Skolem function)

  $\forall x\forall y\exists z\ F(xyz)$ becomes $\forall x\forall y\ F(xy\mathbf{Sk(xy)})$
- remove $\forall$
- put into CNF, drop operators, standardize apart again; add **unification:**

  1. {Px, Qxy}
  2. {¬Pa, Rbz}
  3. {Qay, Rbz} 1,2

a very modest example

- assume: Art is father of Joe, Bob is father of Kim, fathers are parents. **Is Art parent of Joe?**

  1. {Faj} 2. {Fbk} 3. {¬Fxy, Pxy} 4. **{¬Paj}** neg of conclusion

  5. {Paj} 1,3

  6. {Pbk} 2,3

  7. {¬Faj} 3,4

  8. {} 1,7

  9. {} 4,5

- **who is Joe's parent?** 1..3 as above; 4. **{¬Pzj, Az}**

  5. {Paj} 1,3

  6. {Pbk} 2,3

  7. {¬Fwj,Aw} 3,4

  8. {Aa} 4,5

  9. {Aa} 1,7 A(i.e. answer)=Art

unification --- general pattern matching

- unification is needed to make predicate logical expressions **identical** for resolution
- a substitution σ is a set of assignments of terms to vbles, no vbl being assigned more than 1 term
  - E σ (subst. inst. of E): replace vbls in E by terms assigned by σ.
    vbls in E, not mentioned in σ: unchanged.
    assignments in σ to vbls not in E: ignored.
- $E_1 \sigma = E_2 \sigma$: common instance of E_1, E_2;
  σ is a unifier for E_1 and E_2
- $E_1, \ldots, E_n$ are unifiable if there is a σ making them identical:

  $E_1 \sigma = E_2 \sigma = \ldots.$

unification, cont.

- for all E, $E(\sigma \circ \lambda) = (E\sigma) \circ \lambda$
- σ is a **most general unifier** of set X if every unifier λ of X satisfies $\lambda = \sigma \circ \lambda$
- if X is unifiable then there exists a MGU

p(x), p(5); $\sigma = \{x \leftarrow 5\}$; ci: $p(x)\,\sigma = p(5)\,\sigma = p(5)$.

p(x,x), p(5,y); $\sigma = \{x \leftarrow 5, y \leftarrow 5\}$; ci: p(5,5).

p(1,3), p(x,g(5,y)); no match

p(x,g(joe,y)), p(h(3),g(z,mary))

$\sigma = \{x \leftarrow h(3), y \leftarrow mary, z \leftarrow joe\}$; ci: p(h(3),g(joe,mary)).

p(y,g(jack,y)), p(mary,g(w,z))

$\sigma = \{y \leftarrow mary, w \leftarrow jack, z \leftarrow mary\}$; ci: p(mary, g(jack,mary)).

unification, cont.

- each vbl is associated with at most one expression

- no vbl with an associated expression occurs in any of the assoc exprs
  - e.g. {x/g(y), y/f(x)} not a substitution

- composing σ_1 with σ_2: apply σ_1 to terms of σ_2 and add to σ_2 the bindings from σ_1
  - {w/g(x,y)} ° {x/A, y/B, z/C} = {w/**g(A,B)**, x/A, y/B, z/C}
  - {x/A, y/B, z/C} unifies p(A,y,z) and p(x,B,z) but - instead of z/Z - we could have z/D, z/f(w) or nothing ….

    MGU = {x/A, y/B}

unification algorithm

```
MGU (x, y) ←
    x = y → return {};
    Vbl(x) → return (MGUvar (x,y));  Vbl(y) → return (MGUvar (y,x));
    Constant(x) or Constant(y) → return false; ¬(length(x) = length(y)) → return false;
    i ← 0; g ← { };
    while i < length(x) do
            s ← MGU (Part(x, i), Part(y, i));  // toplevel fun or pred constant is Part 0
            s false → return false;
            g ← Compose(g, s); x ← Subst(x, g); y ← Subst(y, g); // returns expr σ
            i++;
    return g; // i.e. MGU
```

length=# of args; Subst(expr, substitution) returns resulting expr after applying substitution; Part(expr,i) is ith part of expr.

unification algorithm, cont.

MGUvar (x, y) ← // don't unify p(x) with p(f(x)) ….

Includes (x, y) → return false; // fail

return ({x/y}).

Includes (var, expr) checks whether var occurs in term with which it is being unified ….
occurs check

occurs check prevents circularities:

given (not (sees ?z ?z)), (if (not(sees ?x (feet ?x))) (shouldDiet ?x)),

1. call of MGUvar(?z ?x) returns {?z/?x}

2.call: ?z already bound to ?x, thus

3. (recursive) call: MGUvar (?x (feet ?x));

without occurs check, this would return {?x/(feet ?x)}. **not** a unifier! We don't want:
(shouldDiet (feet ?x))!!

unification algorithm, cont.

given:

(on ?x table)

(if (on something ?x)(collapses ?x))

does this imply:

(collapses table) ???

(on ?x table) and (on something ?x) do **not** unify because of **coincidental** occurrence of ?x !

* don't forget to rename vars so that no var occurs in more than one clause: **standardizing apart**

(on ?x table) and (on something ?y) unify: {?x/something, ?y/table}

unification, cont.

- f(X,a)=f(a,X)
  - succeeds with X=a
- likes(jane,X)=likes(X,jim)
  - fails
- f(X,Y)=f(P,P)
  - succeeds with X=P, Y=P
- [a,b | X]=[A,B,c]
  - succeeds with A=a, B=b, X=[c]
- [X | Y]=[a]
  - succeeds with X=a, Y=[]
- [X | Y]=[]
  - fails