Part I
Inference in first-order logic

1 Backward chaining

Substitution and Composition

Definition 1. Given p a sentence and θ_1, θ_2 two substitutions, the *composition* of θ_1 and θ_2 is the substitution $\theta = \text{COMPOSE}(\theta_1, \theta_2)$ such that:

$$\text{SUBST}(\theta, p) = \text{SUBST}(\theta_1, \text{SUBST}(\theta_2, p)) = \text{SUBST}(\theta_2, \text{SUBST}(\theta_1, p))$$

Example 2. Sentence p is $P(y) \wedge Q(x) \Rightarrow R(z)$
Consider $\theta_1 = \{y/Toto, z/Titi\}$, $\theta_2 = \{x/Tata\}$
$\text{SUBST}(\theta_1, p) = P(Toto) \wedge Q(x) \Rightarrow R(Titi)$
$\text{SUBST}(\theta_2, p) = P(y) \wedge Q(Tata) \Rightarrow R(z)$
$\text{SUBST}(\text{COMPOSE}(\theta_1, \theta_2), p) = P(Toto) \wedge Q(Tata) \Rightarrow R(Titi)$

Backward chaining: main idea

Definition 3. Given a definite clause $p_1 \wedge \cdots \wedge p_n \Rightarrow c$, c is called the *Head*. $p_1 \wedge \cdots \wedge p_n$ is called the *Body*.

Goal-driven algorithm.

1. Unification of the goal with the head of a rule
2. Propagation of the substitution to the body. Every premise of the body is a new goal
3. Apply BC recursively on the new goals...

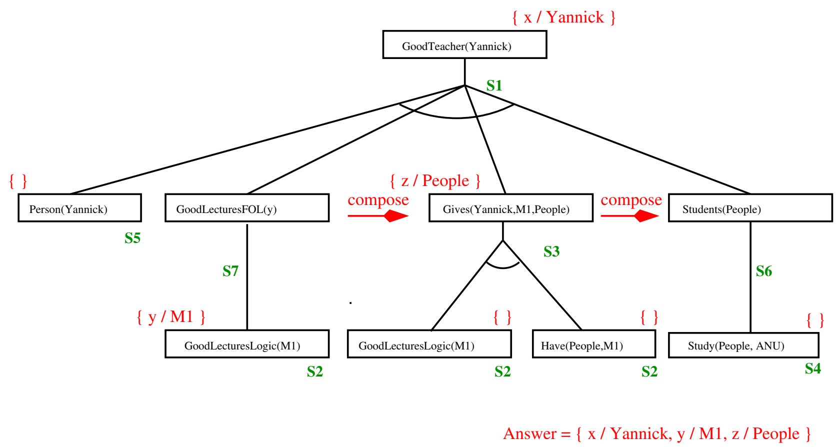
Based on a depth-first search (DFS).

Backward chaining: algorithm

```
function FOL-BC-ASK(KB, goals, θ) returns a set of substitutions
   inputs: KB, a knowledge base
           goals, a list of conjuncts forming a query (θ already applied)
           θ, the current substitution, initially the empty substitution { }
   local variables: answers, a set of substitutions, initially empty

   if goals is empty then return {θ}
   q' ← SUBST(θ, FIRST(goals))
   for each sentence r in KB
           where STANDARDIZE-APART(r) = ( p1 ∧ ... ∧ pn ⇒ q)
           and θ' ← UNIFY(q, q') succeeds
       new_goals ← [p1, ..., pn|REST(goals)]
       answers ← FOL-BC-ASK(KB, new_goals, COMPOSE(θ', θ)) ∪ answers
   return answers
```

Backward chaining: DFS-tree



Example 4.

Properties of BC

Depth-first recursive proof search: space is *linear* in size of the proof.

Incomplete due to infinite loops (DFS). To fix that, we have to check the current goal against every goal in the stack.

Inefficient due to repeated subgoals. To fix that we must use a cache of previous results (*memoization*)

So what? If BC is not so good, why do we talk about it? Well, it is widely used and with good optimisations it works! (linear algorithm): PROLOG

2 Resolution

Another Knowledge base

Example 5. Everyone who loves all animals is loved by someone. Anyone who kills an animal is loved by no one. Jack loves all animals. Either Jack or Curiosity killed the cat, who is named Tuna. Did Curiosity kill the cat?

Another Knowledge base

Example 6. "Everyone who loves all animals is loved by someone." $\forall x\ [\forall y\ Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y\ Loves(y,x)]$ "Anyone who kills an animal is loved by no one.." $\forall x[\exists y Animal(y) \wedge Kills(x,y)] \Rightarrow [\forall z\ \neg Loves(z,x)]$ "Jack loves all animals" $\forall x\ Animal(x) \Rightarrow Loves(Jack,x)$ "Either Jack or Curiosity killed the cat, who is named Tuna" $Kills(Jack,Tuna) \vee Kills(Curiosity,Tuna)$ Tuna is a cat $Cat(Tuna)$ A cat is an animal $\forall x\ Cat(x) \Rightarrow Animal(x)$ Question: Did Curiosity kill the cat? $Kills(Curiosity,Tuna)$

Conjunctive Normal Form for FOL

A sentence in a *Conjunctive Normal Form* is a conjunction of clauses, each clause is a disjunction of literals.

Every sentence in FOL (without equality) is logically equivalent to a FOL-CNF sentence.

Example 7. "Everyone who loves all animals is loved by someone" $\forall x\ [\forall y\ Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y\ Loves(y,x)]$ has the following CNF
$[Animal(F(x)) \vee Loves(G(x),x)] \wedge [\neg Loves(x,F(x)) \vee Loves(G(x),x)]$.

Conversion to CNF

1. Elimination of implications
   - $A \Rightarrow B \equiv \neg A \vee B$
2. Move $\neg$ inwards
3. Standardize variables
4. Skolemisation
5. Drop the universal quantifiers
6. Distribute $\vee$ over $\wedge$

Move $\neg$ inwards and variable standardization

$$\neg\forall x\ p \equiv \exists x\ \neg p$$

$$\neg\exists x\ p \equiv \forall x\ \neg p$$

$$(\forall x\ P(x)) \vee (\exists x\ Q(x))$$

x is used twice but it does not represent the same thing (two diffrent scopes). To avoid confusion, we rename:

$$(\forall x\ P(x)) \vee (\exists y\ Q(y))$$

Skolemization

Definition 8. *Skolemisation* is the process of removing existential quantifiers by elimination.

- Simple case = Existential Instanciation
- Complex case = Use of *Skolem functions*

Example 9. Simple case: $\exists x\ P(x)$
Using EI, we have: $P(A)$
Complex case: $\forall x\ [\exists y\ P(x, y)]$
Using EI, we have: $\forall x\ P(x, A)$ *wrong*
Use of a Skolem function $F(x)$: $\forall x\ P(x, F(x))$
(y in is the scope of x)

Conversion to CNF: example

Example 10. $\forall x\ [\forall y\ Animal(y) \Rightarrow Loves(x, y)] \Rightarrow [\exists y\ Loves(y, x)]$

1. Eliminate implications: $\forall x\ [\neg\forall y\ \neg Animal(y) \lor Loves(x, y)] \lor [\exists y\ Loves(y, x)]$
2. Move $\neg$ inwards
    - $\forall x\ [\exists y\ \neg(\neg Animal(y) \lor Loves(x, y))] \lor [\exists y\ Loves(y, x)]$
    - $\forall x\ [\exists y\ \neg\neg Animal(y) \land \neg Loves(x, y)] \lor [\exists y\ Loves(y, x)]$ (De Morgan)
    - $\forall x\ [\exists y\ Animal(y) \land \neg Loves(x, y)] \lor [\exists y\ Loves(y, x)]$ (double negation)
3. Standardize variables: $\forall x\ [\exists y\ Animal(y) \land \neg Loves(x, y)] \lor [\exists z\ Loves(z, x)]$
4. Skolemization: $\forall x\ [Animal(F(x)) \land \neg Loves(x, F(x))] \lor [Loves(G(x), x)]$
5. Drop universal quantifiers: $[Animal(F(x)) \land \neg Loves(x, F(x))] \lor [Loves(G(x), x)]$
6. Distribute $\lor$ over $\land$: $[Animal(F(x)) \lor Loves(G(x), x)] \land [\neg Loves(x, F(x)) \lor Loves(G(x), x)]$

Resolution: inference rule

$$\frac{\ell_1 \lor \cdots \lor \boxed{\ell_i} \lor \cdots \lor \ell_k, \qquad \ell'_1 \lor \cdots \lor \boxed{\ell'_j} \lor \cdots \lor \ell'_n}{\textsc{Subst}(\theta, \ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor \ell'_1 \lor \cdots \lor \ell'_{j-1} \lor \ell'_{j+1} \lor \cdots \lor \ell'_n)}$$

with θ a substitution such that $\textsc{Unify}(\boxed{\ell_i}, \neg\boxed{\ell'_j}) = \theta$

Example 11.

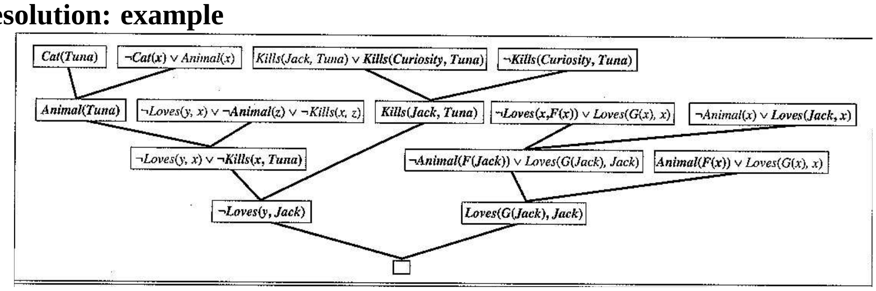
$$\frac{[Animal(F(x)) \lor \boxed{Loves(G(x), x)}] \quad [\boxed{\neg Loves(u, v)} \lor \neg Kills(u, v)]}{Animal(F(x)) \lor \neg Kills(G(x), x)}$$

$\theta = \{u/G(x), v/x\}$

Resolution algorithm

Definition 12. *Proof by contradiction:* given KB, to prove α, we prove that $KB \land \neg\alpha$ is not satisfiable.

Resolution: example



Resolution Proof that Curiosity has killed the cat:

- $\neg\alpha$ is $\neg Kills(Curiosity, Tuna)$
- Use of the factoring rule to infer $Loves(G(Jack), Jack)$

Dealing with equality

There are several ways to deal with $t_1 = t_2$. One of them is *Paramodulation*:

$$\frac{\ell_1 \vee \cdots \vee \ell_k \vee t_1 = t_2, \qquad \ell'_1 \vee \cdots \vee \ell'_n[t_3]}{\textsc{Subst}(\theta, \ell_1 \vee \cdots \vee \ell'_n[y])}$$

where

$$\textsc{Unify}\ (t_1, t_3) = \theta$$

This inference rule can be used during the resolution algorithm.

Example 13.

$$\frac{Father(John) = Father(Richard) \quad Male(Father(x))}{Male(Father(Richard))}$$

$\theta = \{x/John\} = \textsc{Unify}(Father(John), Father(x))$

Theorem provers

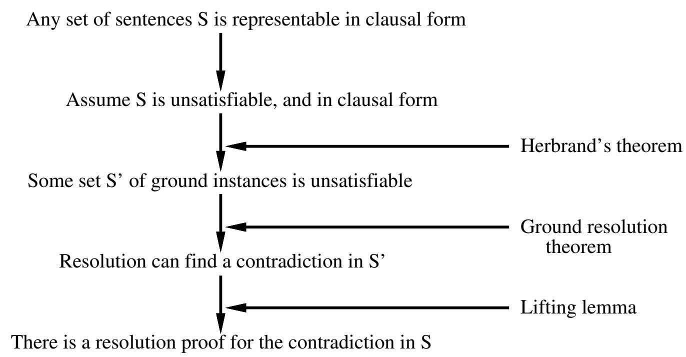
Unlike logic programming language, *Theorem provers* cover FOL (no restriction on Definite Clauses). Their algorithm is based on resolution.

Theorem prover: OTTER

Using the resolution algorithm in a "clever" way.

- *Unit preference*: Inference of sentences with a minimal number of literals (more chance to get the empty clause)
- *Set of support*: What is the set of clauses in KB that will be *useful*?
- *Input resolution*: Always using a sentence from KB or α to apply the resolution rule.
- *Subsumption*: Elimination of sentences that are subsumed by (more specific than) an existing sentence in the KB.

Completeness of resolution


Any set of sentences S is representable in clausal form
Assume S is unsatisfiable, and in clausal form
Herbrand's theorem
Some set S' of ground instances is unsatisfiable
Ground resolution theorem
Resolution can find a contradiction in S'
Lifting lemma
There is a resolution proof for the contradiction in S

Summary

- *Propositionalisation*: very slow
- *Unification* techniques: much more efficient
- Generalised Modus Ponens: FC and BC on Definite clauses
  - FC for *deductive databases*
  - BC for *logic programming*
- Entailement problem is semi-decidable
- Generalised *resolution*: complete proof system (CNF)

To Infinity and Beyond!

Using theorem proving to automatically prove everything...

To prove everything, we need to prove everything in arithmetic...

Logic for arithmetic: 0, $S(..)$, $\times$, $+$, $Expt$ (extension of FOL, more expressive)

Gödel said (after a proof on 30 pages):
"Whatever your logic is, if your logic can express arithmetic, whatever your KB is, I can exhibit a sentence in your logic such that the sentence is entailed by KB but there's no way to prove it by inference thanks to your KB"

Sorry for the inconvenience...