Part I
Inference in first-order logic

1 Backward chaining

Substitution and Composition

Definition 1. Given $p$ a sentence and $\theta_1$, $\theta_2$ two substitutions, the composition of $\theta_1$ and $\theta_2$ 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 $P(y) \land Q(x) \Rightarrow R(z)$

Consider $\theta_1 = \{y/Toto, z/Titi\}$, $\theta_2 = \{x/Tata\}$

$\text{SUBST}(\theta_1, p) = P(Toto) \land Q(x) \Rightarrow R(Titi)$

$\text{SUBST}(\theta_2, p) = P(y) \land Q(Tata) \Rightarrow R(z)$

$\text{SUBST}(\text{COMPOSE}(\theta_1, \theta_2), p) = P(Toto) \land Q(Tata) \Rightarrow R(Titi)$

Backward chaining: main idea

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

```plaintext
function FOL-BC-Ask(KB, goals, $\theta$) returns a set of substitutions
inputs: KB, a knowledge base
        goals, a list of conjuncts forming a query ($\theta$ already applied)
        $\theta$, the current substitution, initially the empty substitution { }
local variables: answers, a set of substitutions, initially empty
if goals is empty then return { }$^{\theta}$
$\theta' \leftarrow \text{SUBST}(\theta, \text{First}(goals))$
for each sentence $r$ in KB where $\text{STANDARDIZE-APART}(r) = (p_1 \land \cdots \land p_n \Rightarrow q)$ and $\theta' \leftarrow \text{UNIFY} (\bar{q}, \bar{q'})$ succeeds
    new_goals$\leftarrow \{p_1, \ldots, p_n, R\text{ET}(goals)\}$
    answers$\leftarrow \text{FOL-BC-Ask}(KB, new\_goals, \text{COMPOSE}(\theta', \theta)) \cup answers$
return answers
```
**Backward chaining: DFS-tree**

![DFS-tree diagram]

**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.” ∀x[∀y Animal(y) ⇒ Loves(x, y)] ⇒ [∃y Loves(y, x)] “Anyone who kills an animal is loved by no one.” ∀x[∃y Animal(y) ∧ Kills(x, y)] ⇒ ∀z ¬Loves(z, x) “Jack loves all animals” ∀x Animal(x) ⇒ Loves(Jack, x) “Either Jack or Curiosity killed the cat, who is named Tuna” Kills(Jack, Tuna) ∨ Kills(Curiosity, Tuna) “Tuna is a cat” Cat(Tuna) A cat is an animal ∀x Cat(x) ⇒ 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” ∀x[∀y Animal(y) ⇒ Loves(x, y)] ⇒ [∃y Loves(y, x)] has the following CNF

\[ \text{Animal}(F(x)) \lor \text{Loves}(G(x), x) \land \neg \text{Loves}(x, F(x)) \lor \text{Loves}(G(x), x). \]
Conversion to CNF

1. Elimination of implications
   - $A \Rightarrow B \equiv \neg A \lor B$

2. Move $\neg$ inwards

3. Standardize variables

4. Skolemisation

5. Drop the universal quantifiers

6. Distribute $\lor$ over $\land$

Move $\neg$ inwards and variable standardization

\[
\neg \forall x \ p \equiv \exists x \ \neg p \\
\neg \exists x \ p \equiv \forall x \ \neg p \\
(\forall x \ P(x)) \lor (\exists x \ Q(x))
\]

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

\[
(\forall x \ P(x)) \lor (\exists y \ Q(y))
\]

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. $\forall x \forall y \text{Animal}(y) \Rightarrow \text{Loves}(x, y) \Rightarrow [\exists y \text{Loves}(y, x)]$

1. Eliminate implications: $\forall x [\neg \forall y \neg \text{Animal}(y) \lor \text{Loves}(x, y)] \lor [\exists y \text{Loves}(y, x)]$

2. Move $\neg$ inwards
   - $\forall x [\exists y \neg \text{Animal}(y) \lor \text{Loves}(x, y)] \lor [\exists y \text{Loves}(y, x)]$ (De Morgan)
   - $\forall x [\exists y \text{Animal}(y) \land \neg \text{Loves}(x, y)] \lor [\exists y \text{Loves}(y, x)]$ (double negation)

3. Standardize variables: $\forall x [\exists y \text{Animal}(y) \land \neg \text{Loves}(x, y)] \lor [\exists z \text{Loves}(z, x)]$

4. Skolemization: $\forall x [\text{Animal}(F(x)) \land \neg \text{Loves}(x, F(x))] \lor [\text{Loves}(G(x), x)]$

5. Drop universal quantifiers: $[\text{Animal}(F(x)) \land \neg \text{Loves}(x, F(x))] \lor [\text{Loves}(G(x), x)]$

6. Distribute $\lor$ over $\land$: $[\text{Animal}(F(x)) \lor \text{Loves}(G(x), x)] \land [\neg \text{Loves}(x, F(x)) \lor \text{Loves}(G(x), x)]$

Resolution: inference rule

$$\quad \ell_1 \lor \cdots \lor \ell_i \lor \cdots \lor \ell_k, \quad \ell'_1 \lor \cdots \lor \ell'_j \lor \cdots \lor \ell'_n$$

$$\text{SUBST}(\theta, \ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell'_{j+1} \lor \cdots \lor \ell'_n)$$

with $\theta$ a substitution such that $\text{UNIFY}(\ell_i, \neg \ell'_j) = \theta$

Example 11.

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

$$\text{Animal}(F(x)) \lor \neg \text{Kills}(G(x), x)$$

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

Resolution algorithm

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

Resolution: example

Resolution Proof that Curiosity has killed the cat:

- $\neg \alpha$ is $\neg \text{Kills}(\text{Curiosity}, \text{Tuna})$
- Use of the factoring rule to infer $\text{Loves}(G(\text{Jack}), \text{Jack})$
Dealing with equality

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

$$\frac{\ell_1 \lor \cdots \lor \ell_k \lor t_1 = t_2, \quad \ell'_1 \lor \cdots \lor \ell'_n [t_3]}{\text{SUBST}(\theta, \ell_1 \lor \cdots \lor \ell'_n [y])}$$

where

$$\text{UNIFY}(t_1, t_3) = \theta$$

This inference rule can be used during the resolution algorithm.

Example 13.

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

$$\theta = \{x/John\} = \text{UNIFY}(\text{Father}(John), \text{Father}(x))$$

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 $\alpha$ 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

Some set $S'$ of ground instances is unsatisfiable

Resolution can find a contradiction in $S'$

There is a resolution proof for the contradiction in $S$
• 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
• Entailment 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(..), ×, +, 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...