Artificial Intelligence

Augustin Cosse.

Fall 2020

October 6, 2021

So far

- Simple reflex, random agents, Utility based, Goal based
- Improvement through Search Methods (uninformed (DFS, BFS), informed (BS, A^*)).
- Logical Reasoning, Propositional logic (including syntax and semantics)
- Propositional inference (Conjunctive Normal Forms, Resolution rules, Resolution Algorithm),
- Horn + definite clauses, Forward and Backward chaining.

This week

- First Order Logic (Part II)
- Inference in first order logic

Reminders

- In Propositional logic every expression is a sentence which represents a fact about the world.
- Recall that first order logic relies on a stronger ontological commitment which postulates that the world consits of Objects, properties.
- FOL makes it possible to define relations (such as bigger than, inside, part of,...) on the objects. Some of the relations are functions (relations in which there is only one possible value for a given input) (e.g. 'father of', 'best friend',...) others are predicates (in this case the output is a Boolean value)

Inference in First Order Logic

- When discussing Propositional logic we considered a number of inference rules including Modus Ponens, And-Elimination, And-Introduction, Or-Elimination, Or-Introduction and Resolution.
- The rules hold for First Order Logic as well, but we will need additional rules to handle complex sentences with quantifiers.

Inference in First Order Logic

- Let us start with the universal quantifier. Suppose that our knowledge base contains the following axiom stating that every greedy king is evil

$$\forall x \; \mathrm{King}(x) \wedge \mathrm{Greedy}(x) \Rightarrow \mathrm{Evil}(x)$$

- From this sentence, it seems permissible to infer the following set of axioms:

$$\begin{aligned}
&\mathrm{King}(\mathrm{John}) \wedge \mathrm{Greedy}(\mathrm{John}) \Rightarrow \mathrm{Evil}(\mathrm{John})\\
&\mathrm{King}(\mathrm{Richard}) \wedge \mathrm{Greedy}(\mathrm{Richard}) \Rightarrow \mathrm{Evil}(\mathrm{Richard})\\
&\mathrm{King}(\mathrm{Father}(\mathrm{John})) \wedge \mathrm{Greedy}\ (\mathrm{Father}(\mathrm{John})) \Rightarrow \mathrm{Evil}(\mathrm{Father}(\mathrm{John}))\\
&\vdots
\end{aligned}$$

Inference in First Order Logic

- The rule of Universal Instantiation (Elimination) (UI) says that we can infer any sentence obtained by substituting (in the universal sentence) a ground term (i.e a term without variable) for the variable.

- To write this first inference rule, we use the notion of substitution. Let $\text{Subst}(\theta, \alpha)$ denote the result of applying the substitution θ to the sentence α.

- The resulting rule reads as

$$\frac{\forall v, \alpha}{\text{Subst}(\{v/g\}, \alpha)}$$

For example, the three sentences derived before were obtained through the substitutions $\{x/\text{John}\}, \{x/\text{Richard}\}$ and $\{x/\text{Father(John)}\}$

Inference in First Order Logic

- In the rule of Existential Instantiation (Elimination), the variable is replaced by a single new constant symbol. The formal statement is the following. For any sentence α, variable v and constant k that does not appear elsewhere in the knowledge base,

$$\frac{\exists v, \alpha}{\mathrm{Subst}(\{v/k\}, \alpha)}$$

- As an example, from the sentence

$$\exists\, x \; \mathrm{Crown}(x) \wedge \mathrm{OnHead}(x, \mathrm{John})$$

  we can infer the sentence

$$\mathrm{Crown}(C_1) \wedge \mathrm{OnHead}(C_1, \mathrm{John})$$

  As long as C_1 does not appear elsewhere in the knowledge base (i.e. C_1 does not represent anything yet)

Inference in First Order Logic

- The existential sentence says that there is some object satisfying a condition and applying the existential instantiation rule just gives a name to that object.

- Whereas universal instantiation can be applied multiple times to produce many different consequences, Existential Instantiation can be applied only once. Then the existential sentence can be discarded.

- As an example, we no longer need $\exists\, x\ \mathrm{Kill}(x, \mathrm{Victim})$ once we have added the sentence $\mathrm{Kill}(\mathrm{Murderer}, \mathrm{Victim})$ to the knowledge base.

- Stricly speaking the new knowledge base is not equivalent to the old one but it can be shown to be Inferentially equivalent (in the sense that it is satisfiable exactly when the original knowledge base is satisfiable).

Inference in First Order Logic

- Once we have the rule for inferring non quantifier sentences from quantified ones, it becomes possible to reduce first order inference to Propositional inference
- The first idea is that just as an existentially quantified sentence can be replaced by one instantiation, a universally quantifier sentence can be replaced by the set of all possible instantiations. For example, suppose our knowledge base contains just the sentences

$$\forall\, x \ \mathrm{King}(x) \land \mathrm{Greedy}(x) \Rightarrow \mathrm{Evil}\ (x)$$
$$\mathrm{King(John)}$$
$$\mathrm{Greedy(John)}$$
$$\mathrm{Brother(Richard, John)}$$

Inference in First Order Logic

- Then we can apply UI to the first sentence using all possible ground term substitutions from the vocabulary of the knowledge base – in this case $\{x/\text{John}\}$ and $\{x/\text{Richard}\}$. We then obtain

$$\text{King}(\text{John}) \wedge \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})$$
$$\text{King}(\text{Richard}) \wedge \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard})$$

- And we can discard the universally quantifier sentence.

- This technique of propositionalization can be made completely general. That is every first order knowledge base can be propositionalized in such a way that entailment is preserved.

Inference in First Order Logic

- There is however a problem: When our knowledge base contains a function symbol, then infinitely many nested terms such as Father(Father(Father(Father(John)))) can be constructed.
- The propositional algorithms will have difficulty with an infinitely large set of sentences
- Fortunately there is famous theorem by Jacques Herbrand (1930) which states that if a sentence is entailed by the original, first order knowledge base, then there is a proof involving just a finite subset of the propositionalized knowledge base.

Inference in First Order Logic

- Since any such subset has a finite depth of nesting among its gound terms, we can find the subset by first generating all the instantiations with constant symbols (Richard and John) then all the terms of depth 1: Father(Richard) and Father (John) then all terms of depth 2 and so on, until we are able to construct a propositional proof of the entailed sentence.
- The approach that we have sketched via propositionalization is complete. That is any entailed sentence can be proved.
- This is in fact puzzling since the space of possible models is infinite. In other words, if the sentence is not entailed by the KB we might never know it and keep believing with a proof method that keeps running forever.

Inference in First Order Logic

- In First Order Logic, it turns out that we will not know whether a sentence is entailed by the KB until the proof method has converged, which might never happen (e.g. if the KB does not specify anything about the expression)
- The proof procedure can go on and on, generating more and more deeply nested terms, but we will not know whether it is stuck in a hopeless loop or whether the proof is just about to pop up.
- Alan Turing and Alonzo Church both proved (1936) that the question of entailment for First Order Logic is semidecidable - That is, algorithms exist that say yes to every entailed sentence, but no algorithm exists that also says no to every nonentailed sentence.

Inference in First Order Logic

- There still remains some inefficiency in the propositionalization approach we have discussed so far.

- As an example, given the query Evil(x) for $x = \text{John}$ and the KB given by

$$\forall\, x\ \text{King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}\ (x)$$
$$\text{King}(\text{John})$$
$$\text{Greedy}(\text{John})$$
$$\text{Brother}(\text{Richard}, \text{John})$$

- It seems excessive to generate sentences such as

$$\text{King}(\text{Richard}) \wedge \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard})$$

(I.e. since the premisses do not belong to the KB, such implications are meaningless)

Inference in First Order Logic

- The Inference Evil(John) from the set of sentences

$$\forall\, x \; \mathrm{King}(x) \wedge \mathrm{Greedy}(x) \Rightarrow \mathrm{Evil}(x)$$
$$\mathrm{King}(\mathrm{John})$$
$$\mathrm{Greedy}(\mathrm{John})$$

  seems completely obvious

Inference in First Order Logic

- Intuitively we would like to say that if there is a particular substitution θ that makes each of the premises identical to sentences already in the knowledge base, then we can assert the result of the implication after applying θ

- Now assume that instead of having Greedy(John) in the KB, we had $\forall\ y$ Greedy(y). We would still want to be able to infer Evil(John).

- The resulting idea is known as Generalized Modus Ponens. For atomic sentences p_i, p_i' and q, where there is a substitution θ such that $\text{Subst}(\theta, p_i') = \text{Subst}(\theta, p_i)$, for all i, we can write

$$\frac{p_1', p_2', \ldots p_n',\ (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{\text{Subst}(\theta, q)}$$

In the example above $p_1' = \text{King(John)}$, p_2' is Greedy(y), p_1 is King(x) and p_2 is Greedy(x). θ is $\{x/\text{John}, y/\text{John}\}$

Inference in First Order Logic

- We say that Generalized Modus Ponens is a Lifted version of Modus Ponens. It raises Modus Ponens from ground (i.e. variable free) propositional logic to first order logic.
- Lifted inference rules require finding substitutions that make different logical expressions look identical.
- This process is called Unification.
- The `Unify` algorithm takes two sentences and return a unifier for them if one exists.

$$\texttt{Unify(p,q)} = \theta \text{ where } \mathsf{Subst}(\theta, p) = \mathsf{Subst}(\theta, q)$$

Inference in First Order Logic

- Let us consider a particular application of the Unification Algorithm. Suppose that our query is to find all the acquaintances of John. I.e. we want to find all the x's for which the sentence $\text{Knows}(\text{John}, x)$ will return True.

- An answer to this query can be obtain by finding all sentences in the knowledge base that unify with $\text{Knows}(\text{John}, x)$. Examples could include

$$\begin{aligned}
&\texttt{Unify}(\text{Knows}(\text{John}, x), \text{Knows}(\text{John}, \text{Jane})) = \{x/\text{Jane}\} \\
&\texttt{Unify}(\text{Knows}(\text{John}, x), \text{Knows}(y, \text{Bill})) = \{x/\text{Bill}, y/\text{John}\} \\
&\texttt{Unify}(\text{Knows}(\text{John}, x), \text{Knows}(y, \text{Mother}(y))) \\
&\qquad\qquad\qquad\qquad = \{y/\text{John}, x/\text{Mother}(\text{John})\} \\
&\texttt{Unify}(\text{Knows}(\text{John}, x), \text{Knows}(x, \text{Elizabeth})) = \textit{fails}
\end{aligned}$$

Inference in First Order Logic

- The sentence

$$\texttt{Unify}(\text{Knows}(\text{John}, x), \text{Knows}(x, \text{Elizabeth}))$$

  fails as it is not possible to simultaneously give the value John and Elisabeth to x. However, you should recall that $\text{Knows}(x, \text{Elizabeth})$ really stands for 'Everybody knows Elizabeth'.

- We should therefore be able to show that John knows Elisabeth.

- The Misunderstanding arises because the two sentences use the same variable x. It can be avoided by standardizing apart one of the sentences being unified, which means renaming its variables to avoid name clashes.

Inference in First Order Logic

- As an example, we could rename the variable x in $\text{Knows}(x, \text{Elisabeth})$ to x_{17} (without changing the meaning of the sentence). We can then update the outcome of the unification as

$$\begin{aligned}\texttt{Unify}(\text{Knows}(\text{John}, x), \text{Knows}(x_{17}, \text{Elisabeth}))\\ = \{x/\text{Elizabeth}, x_{17}/\text{John}\}\end{aligned}$$

- There is one more difficulty. We said that `Unify` should return a substitution that makes the two arguments look the same. What if there are more than one such argument?

- As an example, consider the following call to the unify function

$$\texttt{Unify}(\text{Knows}\ (\text{John}, x), \text{Knows}(y, z))$$

- Valid substitutions for this call could give $\{y/\text{John}, x/z\}$ but also $\{y/\text{John}, x/\text{John}, z/\text{John}\}$

Inference in First Order Logic

- The first unifier would give $\text{Knows}(\text{John}, z)$ as the result of the unification whether the second unifier would give $\text{Knows}(\text{John}, \text{John})$

- In fact the second result could be obtained from the first through the additional substitution $\{z/\text{John}\}$.

- We say that the first unifier is more general than the second because it places fewer restrictions on the values of the variables.

- It turns out that for every unifiable pair of expressions there is a single Most General Unifier (MGU) that is unique up to renaming and substitution of the variables. For example $\{x/\text{John}\}$ and $\{y/\text{John}\}$ are considered equivalent and so are $\{x/\text{John}, y/\text{John}\}$ and $\{x/\text{John}, y/x\}$.

Inference in First Order Logic

- In the case of the example

$$\texttt{Unify}(\text{Knows}\ (\text{John}, x), \text{Knows}(y, z)),$$

it turns out that the MGU is $\{y/\text{John}, x/z\}$

Function `Unify`(x, y, θ):

input : x variable, constant, list, or compound expression
y, a variable, constant, list, or compound expression
θ, the substitution built up so far (default empty)

if $\theta = \mathit{failure}$ **then**
  return failure
end
else if *is_Variable*(x) **then**
  return `Unify-Var`(x, y, θ)
end
else if *is_Variable*(y) **then**
  return `Unify-Var`(y, x, θ)
end
else if *is_Compound*(x) *and is_Compound*(y) **then**
  return Unify(x.Args, y.Args, Unify(x.*Op*, y.*Op*, θ))
end
else if *is_List*(x) *and is_List(y)* **then**
  return Unifty(x.*Rest*, y.*Rest*, *Unify*(x.*First*, y.*First*, θ))
end
else
  return failure
end

```
Function Unify-Var(var, x, θ):
    if {var/val} ∈ θ then
        return Unify(val, x, θ)
    end
    else if {x/val} ∈ θ then
        return Unify(var, val, θ)
    end
    else if Occur-check(var, x) then
        return failure
    end
    else
        return add {var/x} to θ
    end
```

(The call to x.Op y.Op compares the operators $F(\cdot)$ and $G(\cdot)$ appearing in x and y. The only possibility for a unification to exist is for the two functions to be the same.)

Unification

- In a compound expression such as $F(A, B)$ the `Op` field picks out the function F and the `Args` field picks out the argument list (A, B)
- A robust unification algorithm uses the `Occur-check` function, which ensures that a logic variable is not bound to a structure that contains itself such as in $x = f(x)$.
- Not performing the check can cause the unification to go into an infinite loop in some cases.
- On the other hand, performing the occur-check greatly increases the time taken by unification, even in cases that would not require the check.

Unification

- On top of the `Tell` and `Ask` functions used to inform and interrogate the knowledge base, we will now consider the additional function `Fetch`
- `Fetch` is a function that returns all unifiers such that the query q unifies with some sentence in the Knowledge base.
- The simplest way to implement the function `Fetch` is to combine it with a `Store` routine which stores all the facts in one long list and unify each query against every element of the list

First Order definite clauses

- Recall that in Propositional logic, we introduced a forward chaining algorithm for Horn clauses.
- The idea was simple: we started with the atomic sentences in the Knowledge base, and apply Modus Ponens in the Forward direction
- First Order definite clauses closely resemble propositional definite clauses
- Those clauses are disjunctions of literals of which exactly one is positive

First Order definite clauses

- A FOL definite clause is either an atomic expression (i.e. predicate symbol followed by parenthesized list of items), or is an implication whose antecedent is a conjunction of positive literals and whose consequent is a single positive literal.

- Examples include

$$\mathsf{King}(x) \wedge \mathsf{Greedy}(x) \Rightarrow \mathsf{Evil}\ (x)$$
$$\mathsf{King}(\mathsf{John})$$
$$\mathsf{Greedy}(y)$$

First Order definite clauses: illustration

- Consider the translation of the following excerpt into First Order Logic:

  The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by John Doe, who is American.

- Let us prove that *John Doe* is a criminal

First Order definite clauses: illustration

- For the first sentence "It is a crime for an american to sell weapons to hostile nations"

  $\text{American}(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \Rightarrow \text{Criminal}(x)$

- "Nono has some missiles" can be first translated to FOL as $\exists x,\ \text{Owns}(\text{Nono}, x) \wedge \text{Missile}(x)$

- It is then transformed into two definite clauses by Existential instantiation:

$$\text{Owns}(\text{Nono}, M_1)$$
$$\text{Missile}(M_1)$$

First Order definite clauses: illustration

- "All of its missiles were sold to it by John Doe"

$$\text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{J. Doe}, x, \text{Nono})$$

- We also need to encode the fact that missiles are weapons

$$\text{Missile}(x) \Rightarrow \text{Weapon}(x)$$

- We must know that an enemy of america counts as "hostile"

$$\text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x)$$

- "Doe, who is American"

$$\text{American}(\text{J. Doe})$$

- And "The country Nono, an enemy of America"

$$\text{Enemy}(\mathit{Nono}, \text{America})$$

First Order definite clauses: illustration

- The Knowledge base that we just created contains no function symbol and is therefore an instance of the class of Datalog knowledge bases
- Datalog is a language that is restricted to first order definite clauses with no function symbols.
- The name 'Datalog' comes from the fact that the language can be used to encode the statements made in relational databases

```
Function FOL_Forward-Chaining(KB, α):
    input : KB, the knowledge base, a set of first order definite clauses
            α, the query, an atomic sentence
    local variables new, the new sentences inferred on each iteration
    while new is not empty do
        new ← {}
        for each rule in KB do
            (p_1 ∧ ... ∧ p_n ⇒ q) ← Standardize-Variables(rule)
            for each θ such that Subst(θ, p_1 ∧ ... ∧ p_n) = Subt(θ, p'_1 ∧ ... ∧ p'_n) for some
             p'_1, ... p'_n in KB do
                q' ← Subst(θ, q)
                if q' does not unify with some sentence already in KB or new then
                    add q' to new
                    φ ← Unify(q', α)
                    if φ is not fail then
                        return φ
                    end
                end
            end
            add new to KB
        end
    end
```

Forward Chaining

- Simple Forward chaining in FOL is relatively similar to Forward Chaining in PL
- The algorithm starts with the known facts, then triggers all the rules whose premises are satisfied, adding their conclusions to the known facts
- The process repeats until the query is answered or no new facts are added. Note that a fact is not new if it is just a renaming of an old fact. E.g. $\text{Likes}(x, \text{IceCream})$, and $\text{Likes}(y, \text{IceCream})$ are renaming of each other
- The function `Standardize-Variable` replaces all the variables in its arguments with the new ones that have not been used before.

Forward Chaining

- Let us consider our crime example.
- Implications sentences are

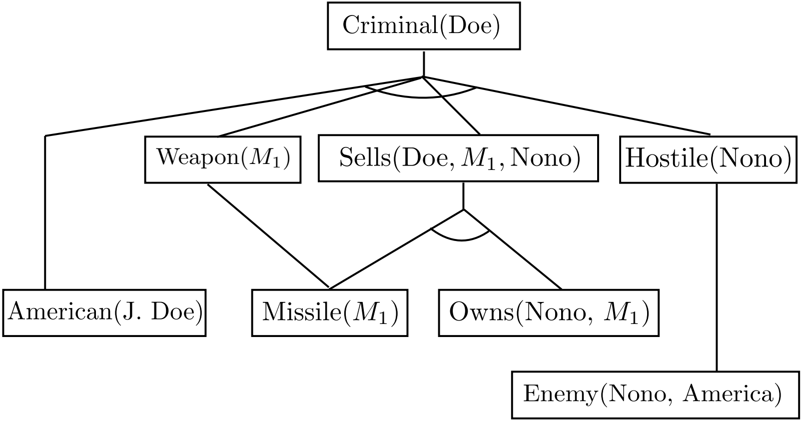
$$\begin{aligned}
&\text{American}(x) \land \text{Weapon}(y) \land \text{Sells}(x, y, z) \land \text{Hostile}(z) \\
&\qquad\qquad\qquad\qquad \Rightarrow \text{Criminal}(x) \\
&\text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{Doe}, x, \text{Nono}) \\
&\text{Missile}(x) \Rightarrow \text{Weapon}(x) \\
&\text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x)
\end{aligned}$$

- Two iterations are required

Forward Chaining

- On the first iteration, the first implication has unsatisfied premises
- The second implication is satisfied with $\{x/M_1\}$ (following from Existential instantiation) and we can add the sentence Sells(Doe, M_1, Nono) to the knowledge base
- The third implication is satisfied with $\{x/M_1\}$ and Weapon(M_1) is added
- Finally the last implication is satisfied with $\{x/\text{Nono}\}$ and Hostile(Nono) can be added.
- On the second iteration, The first rule can be satisfied with $\{x/\text{J. Doe}, y/M_1, z/\text{Nono}\}$

First Order definite clauses: Forward Chaining



Forward Chaining

- On the first iteration, the first implication has unsatisfied premises
- After Forward chaining completed, no new inference is possible for the obtained KB because every sentence that could be obtained by forward chaining is already contained explicitly in the KB
- Such a Knowledge base is called a fixed point of the inference process
- The FOL forward chaining algorithm is sound (if FC derives α then $KB \models \alpha$) because every inference is just an application of Generalized Modus Ponens which is sound.
- The FOL forward chaining algorithm is complete (i.e answers every query whose answers are entailed by the KB) for definite clause knowledge bases

Function `FOL_Backward-Chaining`(*KB, query*):

  return `FOL_BackwardChaining_OR`(KB, query, $\{\}$)

End Function

Function `FOL_BackwardChaining_OR`(*KB, goal,* θ):

  for *each rule* ($lhs \Rightarrow rhs$) *in* `Fetch-Rules-For-Goal`*(KB, goal)* **do**

    (lhs, rhs) $\leftarrow$ `Standardize-Variables`(lhs, rhs)

    for *each* θ' *in* `FOL-BC_AND`*(KB, lhs,* `Unify`($rhs, goal, \theta$) *)* **do**

      yield θ'

    end

  end

End Function

```
Function FOL_BackwardChaining_AND(KB, goals, θ):
    if θ = failure then
        return
    end
    else if length(goals)= 0 then
        yield θ
    end
    else
        first, rest ← First(goals), Rest(goals)
        for each θ′ in FOL-BC-OR(KB, Subst(θ, first), θ) do
            for each θ″ in FOL-BC-AND(KB, rest, θ′) do
                yield θ″
            end
        end
    end
End Function
```

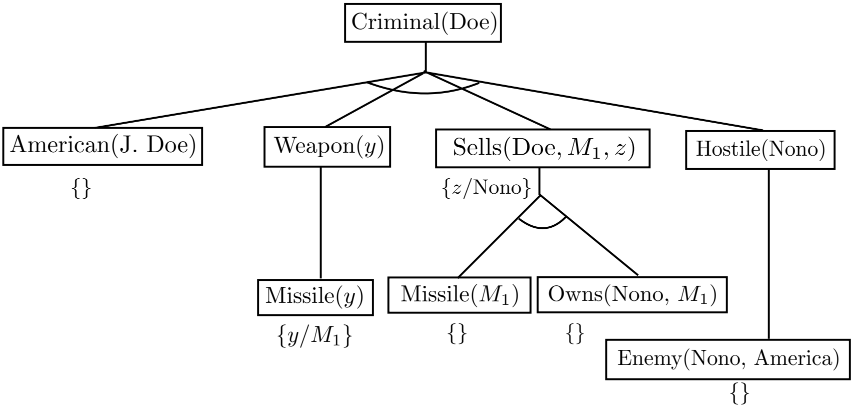
Backward Chaining

- The backward chaining `FOL-BC-ASK(KB,goal)` algorithm returns a proof if the knowledge base contains a clause of the form lhs $\Rightarrow$ goal. where lhs is a list of conjuncts.
- An atomic fact like American(Doe) is considered a clause whose lhs is the empty list.
- Note that a query that contains variables might be proved in multiple ways. For example, the query Person(x) (equivalent to find an object satisfying the 'Person' predicate) could be proved with the substitution $\{x/John\}$ and $\{x/Richard\}$. `FOL-BC-Ask` is thus inplemented as a generator

Backward Chaining

- Backward is a kind of And/Or search. The 'Or' part because the goal query can be proved by any rule in the KB. The And part because all the conjuncts on the LHS of a clause must be proved.
- `FOL-BC-Or` fetches all clauses that might unify with the goal. standardizing the variables in the clauses and then if the RHS of the clause does unify with the Goal, proving every conjunct in the LHS using `FOL-BC-And`
- That second function in turn works by proving every conjuncts keeping track of the accumulated substitution

First Order definite clauses: Backward Chaining



First Order definite clauses: Resolution

- The resolution idea from Propositional logic can also be extended to First Order Logic as follows
- As in the Propositional case, first order resolution requires the sentences to be in conjunctive normal form
- As an example the sentence

$$\forall\, x \; \text{American}(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \Rightarrow \text{Criminal}(x)$$

becomes the CNF

$$\neg\text{American}(x) \vee \neg\text{Weapon}(y) \vee \neg\text{Sells}(x, y, z) \ldots$$
$$\ldots \vee \neg\text{Hostile}(z) \vee \text{Criminal}(x)$$

First Order definite clauses: Resolution

- The procedure for conversion to CNF is similar to the propositional case. The principal difference arise from the need to eliminate the quantifiers
- As an example, consider the sentence 'Everyone who loves animals is loved by someone'

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

First Order definite clauses: Resolution

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

- Step 1: Eliminate Implications

$$\forall x\, [\neg\forall y \neg \text{Animals}(y) \vee \text{Loves}(x, y)] \wedge [\exists y \text{ Loves}(y, x)]$$

- Step 2. Move $\neg$ inwards. In addition to the rules for negated connectives used in PL, we need rules for negated quantifiers

$$\begin{array}{lll} \neg\forall x\; p & \text{becomes} & \exists x\; \neg p \\ \neg\exists x\; p & \text{becomes} & \forall x, \neg p \end{array}$$

- Using those rules, the sentence above can then read as

$$\forall\, x\, [\exists y\; \neg(\neg \text{Animal}(y) \vee \text{Loves}(x, y))] \vee [\exists\, y \text{ Loves}(y, x)]$$
$$\forall x\, [\exists y\; \neg\neg \text{Animal}(y) \wedge \neg \text{Loves}(x, y)] \vee [\exists y \text{ Loves}(y, x)]$$
$$\forall x\, [\exists y \text{ Animal}(y) \wedge \neg \text{Loves}(x, y)] \vee [\exists y \text{ Loves}(y, x)]$$

First Order definite clauses: Resolution

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

- Step 3. Standardize variables. For sentences like $(\exists x\ P(x)) \vee (\exists x\ Q(x))$ which use the same variable name twice, change the name of one of the variables

$$\forall x\ [\exists y\ \text{Animal}(y) \wedge \neg\text{Loves}(x, y)] \vee [\exists z,\ \text{Loves}(z, x)]$$

- Step 4. Skolemize, Skolemization (removing existential quantifiers). In the simplest case, it follows from the existential Instantiation rule (i.e. translate $\exists x\ P(x)$ into $P(A)$ where A is a new constant).

First Order definite clauses: Resolution

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

- In this case however, we cannot blindly apply Instantiation to our sentence because it does not match the simple pattern $\exists v\ \alpha$. If we blindly applied Instantiation to the two parts of our sentence, we would get

$$\forall x\ [\text{Animal}(A) \wedge \neg\text{Loves}(x, A)] \vee \text{Loves}(B, x)$$

  which has the wrong meaning. I.e. It says that everyone either fails to love a particular animal A or is loved by some particular entity B while the original sentence allows each person to fail to love a different animal or to be loved by a different person.

First Order definite clauses: Resolution

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

- Instead, we want the Skolem entities to depend on x and z

  $$\forall x\; [\text{Animal}(F(x)) \wedge \neg\text{Loves}(x, F(x))] \vee \text{Loves}(G(x), x)$$

  F and G are Skolem functions (Arguments of the Skolem functions are the universally quantified variables)

First Order definite clauses: Resolution

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

- Step 5. Drop universal quantifiers. At this point, all the remaining variables must be universally quantified and all universal quantifiers have been moved to the left. We can therefore just drop the universal quantifier

$$[\text{Animal}(F(x)) \land \neg\text{Loves}(x, F(x))] \lor \text{Loves}(G(x), x)$$

- Step 6. Finally, just as in Propositional Logic, we distributed $\lor$ over $\land$

$$[\text{Animal}(F(x)) \lor \text{Loves}(G(z), x)] \land [\neg\text{Loves}(x, F(x)) \lor \text{Loves}(G(x), x)]$$

First Order definite clauses: Resolution

- Finally, once all the sentences have been translated to Conjunctive Normal Form, The first order resolution rule is simply a lifted version of the propositional resolution rule.
- Two clauses, which are assumed to be standardized apart so that they share no variables can be resolved if they contain complementary literals
- Propositional literals are complementary if one is the negation of the other.
- First order literals are complementary if one unifies with the negation of the other.

First Order definite clauses: Resolution

- We thus have

$$\frac{\ell_1 \vee \ldots \vee \ell_k, \quad m_1 \vee \ldots \vee m_n}{\text{Subst}(\theta, \ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee m_n)}$$

  where $\text{Unify}(\ell_i, \neg m_j) = \theta$

- For example, we can resolve the two clauses

$$\begin{aligned} &[\text{Animals}(F(x)) \vee \text{Loves}(G(x), x)]\,, \\ \text{and} \quad &[\neg\text{Loves}(u, v) \vee \neg\text{Kills}(u, v)] \end{aligned}$$

  by eliminating the complementary literals $\text{Loves}(G(x), x)$ and $\neg\text{Loves}(u, v)$, with unifier $\theta = \{u/G(x), v/x\}$, to produce the resolvent clause

$$[\text{Animal}(F(x)) \vee \neg\text{Killls}(G(x), x)]$$