First Order Logic
Version 1.0

August 18, 2011

1 Why FoL?

Unlike propositional logic (PL) where a propositional variable was either `true` or `false` we now wish to talk about properties that hold about individuals of a domain. So the syntax of FoL has two parts. The first part is a language of *terms* which refers to or picks out either single individuals or sets of individuals of the domain. The second part of the language is akin to PL which expresses properties about the domain of individuals.
Consider the following example:
All IITK students are smart. While this statement can be treated as a proposition the natural way to understand it is to assume that in the set of all people if we pick out an individual who is an IITK student then she/he is smart. Therefore, we need a way to state properties of individuals. This is done using predicates. These are like functions but they take terms that represent individuals of the universe in question as argument and evaluate to `true` or `false`. For example $IITKStudent(x)$ is `true` exactly when x is an IITK student else it is false.$IITKStudent$ is a predicate with a single argument. An example of a binary predicate is $brother(x, y)$. This is `true` exactly when x is y's brother else it is `false`. For yet another example consider:$between(x, y, z)$. This is `true` exactly when y is in between x and z. $between(2, 3, 4)$ is `true` while $between(10, 12, 11)$ is `false`. Here are a few examples and their translation into FoL.

In English	**Logical wff**
All IITK students are smart	$\forall x.(IITKStudent(x) \rightarrow Smart(x))$
There is an IITK student	$\exists x.IITKStudent(x)$
There is a smart IITK student	$\exists x.(IITKStudent(x) \wedge Smart(x))$
Every student is a friend of some student	$\forall x.(Student(x) \rightarrow \exists y.(Student(y) \wedge Friend(x, y)))$
Every student is a friend of some other student	$\forall x.(Student(x) \rightarrow \exists y.(Student(y) \wedge Friend(x, y) \wedge x \neq y))$
There is a student who is a friend of every other student	$\exists x.(Student(x) \wedge \forall y.((Student(y) \wedge x \neq y) \rightarrow Friend(y, x)))$
No student is a friend of Ram	$\neg\exists x.(Student(x) \wedge Friend(x, Ram))$
Ram has at least one sister	$\exists x.Sister(Ram, x)$
Ram has at most one sister	$\forall xy.((Sister(Ram, x) \wedge Sister(Ram, y)) \rightarrow (x = y))$
Ram has exactly one sister	$\exists x.(Sister(Ram, x) \wedge \forall y.(Sister(Ram, y) \rightarrow (x = y)))$

2 The language of FoL

The alphabet of FoL consists of:

- $\mathcal{C}$: a countable set of constant symbols, $a, b, c \ldots$ and subscripted versions of these.
- $\mathcal{V}$: a countable set of variable symbols, $x, y, z \ldots$ and subscripted versions of these.
- $\mathcal{F}$: a countable set of function symbols, $f, g, h \ldots$ and subscripted versions of these, each with a finite arity.

$\mathcal{P}$: a countable set of predicate symbols, $P, Q, R \ldots$ and subscripted versions of these, each with a finite arity.

The logical symbols:$\neg, \vee, \wedge, \rightarrow, \leftrightarrow$.

The quantifier symbols, $\exists$, $\forall$.

Parentheses of different types, comma, period etc.

The language of *terms* is defined as follows:

i) $\forall c \in \mathcal{C}$ c is a term.

ii) $\forall x \in \mathcal{V}$ x is a term.

iii) if $t_1, t_2, \ldots, t_n$ are terms and $f \in \mathcal{F}$ is a function symbol or arity n then $f(t_1, t_2, \ldots, t_n)$ is a term.

iv) The only terms are those formed by the rules i) to iii) above.

An `atomic well formed formula` (awff) is defined as: if $P \in \mathcal{P}$ is an m-ary predicate symbol and $t_1, t_2, \ldots, t_m$ are terms then $P(t_1, t_2, \ldots, t_m)$ is an awff.

A `literal` is an awff or the negation of an awff. That is $P(t_1, t_2, \ldots, t_m)$ or $\neg P(t_1, t_2, \ldots, t_m)$.

A `well formed formula` (wff) of FoL is defined as:

i) Every awff is a wff.

ii) if α_1, α_2 are wffs then: $\neg\alpha_1$, $\alpha_1 \vee \alpha_2$, $\alpha_1 \wedge \alpha_2$, $\alpha_1 \rightarrow \alpha_2$ and $\alpha_1 \leftrightarrow \alpha_2$ are wffs.

iii) if $x \in \mathcal{V}$ and α is a wff then $\exists x.\alpha$ and $\forall x.\alpha$ are wffs.

iv) The only wffs are those formed by the rules i) to iii) above.

3 Bound and free variables

We now define bound and free occurrences of a variable.

Definition 1 (Bound, free occurrences)**.** An occurrence of variable x is `bound` in a wff α if there is a well formed sub-formula β in α of the form $\forall x.\beta$ or $\exists x.\beta$.

An occurrence of a variable x in a wff is `free` if it is not bound.
◁

Definition 2 (Free and bound variable)**.** A variable x is `free` in a wff α if it has at least one free occurrence in α. x is said to be `bound` if it is not free.
◁

Example 3. Consider wff: $\alpha = \forall x.({}_1\exists y.({}_2P(x, f(y)) \vee Q(x) \wedge R(z) \vee \forall x.S(x))_2)_1$ then in α all occurrences of x, y are bound the lone occurrence of z is free. However, consider the sub-formula $\beta = ({}_1\ldots)_1$ of α, where the occurrences of x in P, Q are free but the occurrence in S is a bound occurrence. So, the same variable may occur bound and free in a wff.

Definition 4 (Closed wff)**.** A wff is `closed` if it has no free variables.
◁

4 Interpretations for FoL

For PL interpretations were just boolean valuations for all the propositional variables in the language. An interpretation for a FoL language is more complex. An interpretation defines an association for the symbols in the language of a FoL to the corresponding entities in the domain so that ultimately one can determine the truth value of a formula by referring to the state of affairs in the domain.

To define an interpretation we define a generic FoL language $\mathcal{L}$ with the alphabet $\mathcal{C} = \{c_1, c_2, \ldots\}$, $\mathcal{F} = \{f_1, f_2, \ldots\}$, $\mathcal{V} = \{x_1, x_2, \ldots\}$, $\mathcal{P} = \{P_1, P_2, \ldots\}$. All other FoL languages are some kind of restrictions of the above language. So, defining an interpretation for $\mathcal{L}$ will allow us to define interpretations for other languages as an instantiation of an interpretation for $\mathcal{L}$.

Definition 5 (Interpretation)**.** The interpretation $\mathcal{I}$ of is defined as follows. Let M be the structure (or model) defined by:

> $\mathcal{D}$: countable domain of individuals.
>
> $f_c : \mathcal{C} \to \mathcal{D}$ a function that maps the constant symbols of $\mathcal{C}$ to individuals in $\mathcal{D}$.
>
> $\mathcal{F}_I$: countable set of functions $f_i^* : \mathcal{D}^n \to \mathcal{D}$, where n is the arity of $f_i^* \in \mathcal{F}_I$.
>
> $\mathcal{R}$: a countable set of relations $R_i^* \subseteq \mathcal{D}^m$, where m is the arity of R_i^*.
>
> Mapping function $F^* : \mathcal{F} \to \mathcal{F}_I$ a mapping function that maps a function symbol $f_i \in \mathcal{F}$ to a domain function $f_j^* \in \mathcal{F}_I$ with the same arity.
>
> Mapping function $R^* : \mathcal{P} \to \mathcal{R}_I$ a mapping function that maps a predicate symbol $P_i \in \mathcal{P}$ to a domain relation $R_j^* \in \mathcal{R}$ with the same arity.

To handle free variables and terms we need a mapping for variables. Let $\mu : \mathcal{V} \to \mathcal{D}$ be a function that maps variables to elements of the domain (handles free variables, when needed). mu can be extended to functions as follows: $f_i(t_1, \ldots, t_n)$ maps to $F^*(f_i)(d_1, \ldots, d_n)$ where each term t_i evaluates to domain value d_i.

The truth value of a wff in $\mathcal{L}$ is inductively determined as follows:

> An atomic wff $P(t_1, \ldots, t_n)$ is `true`, where $t_1, \ldots, t_n$ evaluate respectively to domain elements $d_1, \ldots, d_n \in \mathcal{D}$, exactly when $P(d_1, \ldots, d_n) \in R^*(P(\ldots))$.
>
> For all logical connectives the formula is evaluated as per the truth tables for PL for the corresponding logical connective.
>
> $\exists x.\alpha(x)$ is true iff there is some variable mapping μ such that $\alpha(\mu(x))$ evaluates to `true`.
>
> $\forall x.\alpha(x)$ is true iff it is true in every possible variable mapping μ.

◁

If a wff is closed (i.e. does not have free variables) then its truth/falsity does not depend on the variable mapping.

If interpretation I satisfies a formula α or a set of formulae S we write $I \models \alpha$ or $I \models S$, that is I is a *model* of α or I is a model of S. A formula or set of formulae is *satisfiable* if it has some model else it is *unsatisfiable*.

Example 6. Let: $\mathcal{L} = \{\{\}\{f_1\}, \{P\}\}$ be the language of arithmetic with just one function symbol, one predicate symbol and no constants. Consider two interpretations: $I_1 = (\mathcal{Q}^+, eq, g)$, $I_2 = (\mathcal{R}^+, eq, g)$ where $\mathcal{Q}^+$ is the positive rationals and $\mathcal{R}^+$ is the positive reals, eq is equality over the respective domains and $g(x) = x^2$. Let $R^*(P) = eq$ and $F^*(f_1) = g$ for both interpretations. Consider the wff $\forall x.\exists y.P(x, g(y))$ the formula is true exactly when for all elements of the domain the equation $x = y^2$ has a solution. This is `true` for $\mathcal{R}^+$ but `false` for $\mathcal{Q}^+$.

A formula that is `true` in every interpretation is a *valid* formula or *tautology*. A formula that is `false` in every interpretation is a contradiction or is unsatisfiable.

Definition 7. Logical consequence Let S be a set of FoL wffs. The wff α is a `logical consequence` of S, formally also written as $S \models \alpha$ iff every interpretation I that satisfies S also satisfies α. If $Cons(S)$ is the set of all logical consequences of S:

$\alpha \in Cons(S)$ iff $\forall I. I \models S \rightarrow I \models \alpha$.
$\triangleleft$

5 Conversion to clauses

To be able to use resolution a wff in FoL has to be converted to clausal form. The conversion to clausal form, as for PL, is done by the repeated use of the substitution theorem for FoL. The conversion is done in two steps: i) convert to Prenex normal form (PNF) ii) convert PNF to clausal form. Broadly, the steps to convert to PNF are:

1. Replace $\leftrightarrow$ by $\rightarrow$.
2. Replace $\rightarrow$ using the equivalence $(P() \rightarrow Q()) \leftrightarrow (\neg P() \vee Q())$.
3. Push $\neg$ as far inside as possible using DeMorgan and the equivalences $\exists x.P() \leftrightarrow \neg\forall x.\neg P()$ and $\forall x.P() \leftrightarrow \neg\exists x.\neg P()$.
4. Pull out all quantifiers using equivalences: $(\forall x.P(x) \wedge \forall y.Q(y)) \leftrightarrow \forall x.(P(x) \wedge Q(x))$, $(\exists x.P(x) \vee \exists y.Q(y)) \leftrightarrow \exists x.(P(x) \vee Q(x))$
5. Distribute $\wedge$ over $\vee$ and vice versa to arrive at the PNF form: $Q_1 x_1 \ldots Q_n x_n . C_1 \wedge C_2 \wedge, \ldots \wedge C_m$ where each Q_i is either $\exists$ or $\forall$ and each C_i is a clause, that is a disjunction of literals.

Repeated use of the substitution theorem allows us to conclude:

Theorem 8. *Let S be a set of wffs in FoL and let S_{PNF} be obtained by converting each wff in S to the corresponding PNF then $S \leftrightarrow S_{PNF}$.*

To convert a formula in PNF to clausal form we have to *Skolemize* it. We motivate skolemization through an example. Consider the PNF formula: $\forall x.\exists y.P(x, y)$. Informally, this formula says that for every individual c_1 in the domain we can find another individual c_1' such that $P(c_1, c_1')$. The set of pairs $\{(c_1, c_1'), \ldots, (c_i, c_i'), \ldots\}$ can be thought of as a function, say $f_s(x)$ that calculates the relevant y such that $P(x, y)$. Thus the original formula can be written as $\forall x.P(x, f_s(x))$ where we have been able to remove $\exists$ and replace the corresponding existentially quantified variable y by $f_s(x)$. The symbol f_s is called the Skolem function and should be distinct from all other function symbols in the formula or set of formulae. More generally, if an existentially quantified variable is preceded by m universally quantified variables $x_1, \ldots, x_m$ then it can be replaced by the Skolem function $f_s(x_1, \ldots, x_m)$. This process is called Skolemization. Loewenheim and Skolem proved the following:

Theorem 9 (Loewenheim-Skolem's theorem)**.** *Let S_{PNF} be a set of formulae in PNF. S_{PNF} is satisfiable iff the corresponding skolemized set S_{SK} is satisfiable.*

Note that we do not get an equivalence between the PNF formula and the skolemized formula, only satisfiability.

Exercise: Construct a counter example to show that equivalence does not hold between a PNF formula and its skolemized version.

Since all variables are now universally quantified we simply drop the quantifiers and work with the clauses $C_1 \wedge \ldots \wedge C_n$ that can now be treated as a set $\{C_1, \ldots, C_n\}$ since they are all implicitly connected by $\wedge$. Thus starting from the set S of wffs in FoL we have obtained the set of clauses S_c with the property that S is satisfiable iff S_c is satisfiable (clearly, the same works for unsatisfiability as well).

6 Resolution

Multiple inference systems are available for FoL. The simplest and most widely used system for mechanical theorem proving is resolution. The resolution rule is similar to the one for PL with the requirement of unification (mgu) before resolution.

Definition 10 (Resolution for FoL in clausal form)**.** Let $C_1 = P(t_1, \ldots, t_n) \vee C_1'$ and $C_2 = \neg P(u_1, \ldots, u_n) \vee C_2'$ be two clauses with complementary literal $P()$. If there exists an mgu θ that unifies the set $\{t_1 = u_1, \ldots, t_n = u_n\}$ (or another way to look at it is unifies $P(t_1, \ldots, t_n), P(u_1, \ldots, u_n)$) then the clauses C_1, C_1 can be resolved to yield the resolvent clause $C_r = C_1'\theta \vee C_2'\theta$.
◁

Definition 11 ($\underset{R}{\vdash}$)**.** If S is a set of clauses and the clause T can be derived from $S \cup \mathit{Resolvents}$ by one or more resolution steps then T is derivable from S and we write: $S \underset{R}{\vdash} T$
◁

Theorem 12 (Resolvent is logical consequence)**.** *Let $C_1 = P(t_1, \ldots, t_n) \vee C_1'$ and $C_2 = \neg P(u_1, \ldots, u_n) \vee C_2'$ be two clauses with complementary literal $P()$ and let θ be the mgu. Then the resolvent $C_r = C_1'\theta \vee C_2'\theta$ is a logical consequence of C_1 and C_2. That is: $\{C_1, C_2\} \models C_r$.*

Definition 13 ($\square$ - null clause)**.** $\square$ stands for the contradictory or unsatisfiable clause.
◁

Typically, this is derived by resolving $C_1 = P(t_1, \ldots, t_n)$ with $C_2 = \neg P(u_1, \ldots, u_n)$ where there is unifier θ that unifies $t_1, \ldots, t_n$ with $u_1, \ldots, u_n$ - equivalent of resolving P and $\neg P$ in PL.

Theorem 14 (Soundness)**.** *A set of clauses S is unsatisfiable if $\square \in Cons(S)$. That is: $S \underset{R}{\vdash} \square$ implies S is unsatisfiable.*

Theorem 15 (Completeness)**.** *If S is an unsatisfiable set of clauses then $\square$ is derivable from S. That is: S is unsatisfiable implies $S \underset{R}{\vdash} \square$*

Let S be a set of clauses, the theory, and T a theorem that we wish to prove from the theory S. To prove T we construct the set $S_T = S \cup \{\neg T\}_c$, where $\{\neg T\}_c$ represents the negation of the theorem in clausal form, and derive the $\square$ clause after adding $\{\neg T\}_c$ to S. If T is a logical consequence of S then S_T will be unsatisifiable and theorem 15 guarantees that we can derive the $\square$ clause by resolution.

7 Substitutions and unifiers

Definition 16 (Substitution)**.** A `substitution` $\theta = \{t_1|x_1, \ldots, t_n|x_n\}$ where $t_1, \ldots, t_n$ are terms $x_1, \ldots x_n \in \mathcal{V}$ are distinct variables.

If ϕ is a term then $\phi\theta$ denotes the term obtained by simultaneously substituting $x_1, \ldots, x_n$ by $t_1, \ldots, t_n$ respectively in ϕ.
◁

Definition 17 (Composition)**.** Two substitutions $\theta_1 = \{t_1|x_1, \ldots, t_n|x_n\}$, $\theta_2 = \{s_1|y_1, \ldots, s_m|y_m\}$ can be `composed` as $\theta_1 \circ \theta_2$ (also written $\theta_1\theta_2$) and is the substitution:
$\theta = \{t_1\theta_2|x_1, \ldots, t_n\theta_2|x_n, s_1|y_1, \ldots, s_m|y_m\} - (\{t_i\theta_2|x_i \mid x_i = t_i\theta_2\} \cup \{s_i|y_i \mid y_i \in \{x_1, \ldots, x_n\})$.
◁

Example 18. Let $\theta_1 = \{f(y)|x, z|y\}$, $\theta_2 = \{a|x, b|y, y|z\}$ then $\theta = \theta_1 \circ \theta_2 = \{f(b)|x, y|y, a|x, b|y, y|z\} - (\{y|y\} \cup \{a|x, b|y\}) = \{f(b)|x, y|z\}$.

Theorem 19 (Associativity of composition)**.** *Composition of substitutions is associative. That is if* $\theta_1,\ \theta_2, \theta_3$ *are substitutions then* $(\theta_1 \circ \theta_2) \circ \theta_3 = \theta_1 \circ (\theta_2 \circ \theta_3)$

Definition 20 (Unifier)**.** Given a set of terms $\tau = \{t_1, \ldots, t_n\}$ a `unifier` for τ is a substitution θ for variables in τ such that $\tau\theta$ is a singleton set. That is $t_1\theta = t_2\theta = \ldots = t_n\theta$. The set τ is said to be `unifiable` if such a unifier exists else it is `not unifiable`.
◁

Example 21. Let $\tau = \{f(x_1, h(x_1), x_2), f(g(x_3), x_4, x_3)$ then one substitution that unifies τ is $\{g(x_3)|x_1, h(g(x_3))|x_4, x_3|x_2\}$. Another substitution that unifies is $\{g(a)|x_1, h(g(a))|x_4, a|x_2\}$, where a is some constant.

The above example motivates the definition of the most general unifier (mgu).

Definition 22 (Most general unifier (mgu))**.** A unifier θ for a set of terms τ is a `most general unifer` if every other unifier γ of τ can be written as: $\gamma = \theta \circ \chi$ where χ is some substitution.
◁

In example 21 the mgu is $\{g(x_3)|x_1, h(g(x_3))|x_4, x_3|x_2\}$ and the second unifier can be obtained by composing the mgu with the substitution $\{a|x_3\}$.

7.1 Algorithms for unification

We look at two algorithms below. In each algorithm we restrict ourselves to finding the unifier for two terms - since this case is the one encountered in resolution theorem proving.

Definition 23 (Disagreement set)**.** Given two terms $t_1,\ t_2$ the `disagreement set` is the set of subterms at which the terms t_1 and t_2 first disagree textually. ◁

Example 24. If we look at the two terms in example 21 $t_1 = f(x_1, h(x_1), x_2)$, $t_2 = f(g(x_3), x_4, x_3)$ and the first point at which they disagree are the characters x in t_1 and g in t_2. The set of subterms at that point are: $\{x_1, g(x_3)\}$ which is the disagreement set.

The first unification algorithm uses the disagreement set iteratively to find the mgu.

Algorithm 7.1: UNIFY(t_1, t_2)

output (mgu if t_1, t_2 are unifiable; otherwise ϕ the null unifier)
$\sigma \leftarrow \phi$
while $(t_1 \neq t_2)$
  do
  $ds \leftarrow calcDisagreementSet(t_1, t_2)$
  if (ds contains variable x and term t and x does not occur in t)
    then $\sigma \leftarrow \sigma \circ \{t|x\}$
    else return (ϕ) **comment:** not unifiable
  $t_1 \leftarrow t_1\sigma$
  $t_2 \leftarrow t_2\sigma$
return (σ)

The second algorithm due to [1] finds the mgu by setting up a set of equations and iteratively modifying this set non-deterministically using a set of rules. The mgu is the most general substitution θ such that $t_1\theta = t_2\theta$. We start with the equation $t_1 = t_2$ and apply two transformations that produce equivalent (i.e. with the same unifer, if it exists) sets of equations.

Term reduction if we have the equation $f(t_1, \ldots, t_n) = f(u_1, \ldots, u_n)$ then we replace this equation by the n equations $t_1 = u_1, \ldots, t_n = u_n$. If n is 0 then it is a constant and it is just erased.

Variable elimination If the equation has the form $x = t$ where x is a variable and t a term then the new set of equations is obtained by applying the substitution $\{t|x\}$ for both sides of all other equations. The original equation is retained in the set.

The following two theorems hold:

Theorem 25. *Let $f_1(t_1, \ldots, t_n) = f_2(u_1, \ldots, u_m) \in \tau$ be an equation in the set of equations τ. if $f_1 \neq f_2$ then τ is not unifiable, otherwise the new set τ' obtained by applying term reduction to the equation above is equivalent to τ.*

Theorem 26. *Let $x = t \in \tau$ be an equation in the set of equations τ. If t is not x and x occurs in t then τ is not unifiable, else the new set τ' obtained by applying variable elimination to τ is equivalent to τ.*

Definition 27 (Equations in solved form)**.** A set of equations is in `solved form` iff it satisfies:

1. Each equation is of the form $x_i = t_i$, $i = 1, \ldots m$.
2. Every variable occurring in the lhs of an equation occurs only there.

◁

An equation set in solved form produces the unifier $\theta = \{t_1|x_1, \ldots, t_m|x_m\}$. Any other unifier β can be obtained by composition $\beta = \theta \circ \chi$ where χ is some substitution.

The algorithm below finds a unifier using the following equational transformations non-deterministically - the first two are purely syntactic. The transforms are:

A Replace any equation of the form $t = x$ by $x = t$.

B Erase any equation of the form $x = x$.

C For equation $t_1 = t_2$, t_1, t_2 are not variables, if the function symbols in t_1, t_2 are different return ϕ (failure), else apply term reduction.

D Choose an equation of the form $x = t$ where $t \neq x$ and x occurs in at least one other equation. If x occurs in t return ϕ (failure), else apply variable elimination.

Algorithm 7.2: UNIFY(S)

comment: S is a set of two terms
output (mgu if S is unifiable, else ϕ the null unifier for failure)
while (Some transformation A...D is applicable)
  do
{ Non-deterministically choose an applicable transform
  A...D and apply it.
return (Convert equations S to unifier and return unifier)

References

[1] A Martelli, U Montanari, *An Efficient Unification Algorithm*, ACM ToPLAS, 4(2), 258-282, 1982.