First Order Boolean Logic, part 1

“Logic” is an immense area of human knowledge; I have no plans to even mention all the aspects, and focus here only on simple notions and language, reasonable enough to be used in our daily practice as software developers. Lawyers or philosophers or physicist or linguist will probably need a very different kind of logic; we just won’t even look in those directions.

The Language of First Order Logic

Before talking about logic, even vaguely, we introduce a language that we will use throughout this text. It is neither loose nor strict, but somewhere in the middle.

Definition 1. A first-order language consists of the following:

- *Names* that are supposed to denote some object from some kind of universe (like John denotes a certain human being in a small world where it does it uniquely, or 10 denotes a number, if we know exactly which numbers we are talking about);
- *Functions* that are used to form terms out of names and other terms; this includes infix notation, like 2+3; a function takes its parameters and has a result value;
- *Terms* that are essentially expressions built out of names and functions;
- *n-ary relationships*, which denote the fact that some terms have certain properties see examples below);
- *formulas*, which are expressions built from n-ary relationships applied to terms.

We could write all this in Backus normal form, but probably there is no need.
Summing up, terms are built from names and functions, using parentheses; formulas that use terms; and that’s it so far.

Examples
sin(ln(2.718285)) < cos(exp(0.001))
phoneNumber(John) = “314 159 2654”
Alaska

Note that “first order language” is not something universal; on the contrary, you just come up with a bunch of names, functions, relationships – and voilà, you have a language.

Example 1
Take integer numbers, operations on them, comparisons and equality relationships.

Example 2

Planar geometry. We will need numbers, symbols for points, lines, circles and angles; then we add functions and relationships:

- line(L1, P1, P2) – L1 is a line that contains points P1 and P2
- circle(P, R) – this term denotes a circle with center at P and radius R
- center(C) – this term denotes a point that is center of circle C
- liesOn(P1, L1) – point P1 lies on line L1
- liesOn(P1, C1) – point P1 lies on circle C1
- between(P1, P2, P3) – points P1, P2, P3 are on the same line, and P1 is between P2 and P3
- parallel(L1, L2) – lines L1 and L2 are parallel

Example 3
A version of *naïve set theory* can also be expressed as first-order language:

1. Names denote sets or elements (may not be sets)
2. a ∈ b, where a is an element, b must be a set
3. a = b
4. {a1,...an} is a set, where a1...an are elements
5. s1 ∩ s2 is a set, where s1 and s2 are sets
6. s1 ∪ s2 is a set, where s1 and s2 are sets

Example 4
Directed graphs. We have nodes and edges, and the only relationship:
between(Node1, Node2, Edge3).

Logical Operations

The logical formulas we had before (just relationships/predicates) are actually called *atomic formulas*. We can build more formulas by combining them.

Negation

Having a formula P, its *negation* is denoted as ¬P, and is true if and only if P is false. It is clear from the definition that double negation of a formula P is true if and only if P is true.

Conjunction

Having two formulas, P and Q, their *conjunction* is denoted as P∧Q, and is true if and only if both P and Q are true.

Disjunction

Having two formulas, P and Q, their *disjunction* is denoted as P∨Q, and is true if and only if at least one of P and Q is true.

Sentences: Combining Operations

Out of atomic formulas we build *sentences*, using negation, conjunction and disjunction; to avoid disambiguation, we also use parentheses.

Example
((x < 7)∨((y < x)∧¬isEmpty(”hello world”)))

Properties of Operations

Describing these properties, we use symbol ≡, which vaguely means that the expressions on the left and on the right are equivalent. The exact meaning of such equivalence may vary from theory to theory.

Associativity

Both conjunction and disjunction are associative
(P∨Q)∨R ≡ P∨(Q∨R)
(P∧Q)∧R ≡ P∧(Q∧R)

Commutativity

Both conjunction and disjunction are commutative
P∨Q ≡ Q∨P
P∧Q ≡ Q∧P

Idempotence

Both conjunction and disjunction are idempotent
P∨P ≡ P
P∧P ≡ P

Double Negation

¬¬P ≡ P

DeMorgan Laws

¬(P∨Q) ≡ ¬P∧¬R
¬(P∧Q) ≡ ¬P∨¬R

More Operations

We saw negation, conjunction and disjunction; can we have other operations?

Negation is a unary operation; conjunction and disjunction are binary. How about other arities?

We can start with zero arity, that is, with *constants*, and remember two constants that were always lurking in the background – True and False. There may be more nullary operations, but for the sake of this discussion let us limit ourselves with just two; we need both. The first one, True, is a neutral element for conjunction; the second one, False, is a neutral element for disjunction.

Now take a look at the unary operations. How many can we define on two logical constants? Obviously just 4: here's the table.

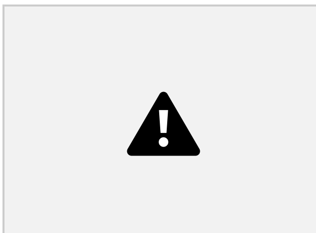
	identity	**always True**	**always False**	**negation**
True	True	True	False	False
False	False	True	False	True

For binary operations there's more combinations (any idea how many?); we will not list them all here, just mention the important ones.

x	**y**	**x∧y**	**x∨y**	**x→y (implication)**	**x↔y (equivalence)**	**x↓y (Pierce arrow, NOR)**	**x↑y, (Sheffer stroke, NAND)**
T	T	T	T	T	T	F	F
T	F	F	T	F	F	F	T
F	T	F	T	T	F	F	T
F	F	F	F	T	T	T	T

You see conjunction and disjunction; implication can be defined as $\neg x \lor y$; equivalence can be defined as $(x \land y) \lor (\neg x \land \neg y)$; actually, any formula has a standard representation via negation, conjunction and disjunction. But there are two operations each of which can be used to represent all other operations. One of them is called *Pierce Arrow*, or *NOR* (because it is equivalent to $\neg(x \wedge y)$)); the other is called Sheffer Stroke, or *NAND* (because it is equivalent to $\neg(x \vee y)$)).

Imagine you have to use just one circuit to build a logical schema; which one would you use? You have two choices, either take NAND, or NOR.



Proving Something

Premises and Conclusions

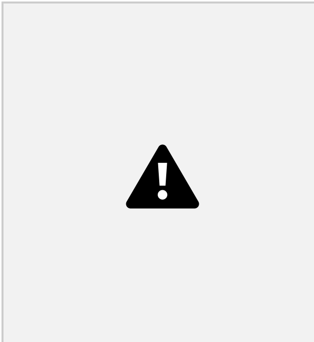
When we have first-order language, we can connect our formulas (see definition), calling some of them *premises* and some – *conclusions*. Informally, you list premises, and then come up with a conclusion, or go in the opposite direction: conclusions, because we have premise1, premise2, etc. Nobody said we have to do it right.

Example 1

“All men are mortal; Superman is a man, *hence* Superman is mortal.”
Some may agree with this, some may disagree; some will ask “which Superman”, thus invalidating the whole discussion.

Example 2

“Pavlova is a man: after all, Pavlova is mortal, and all men are mortal”. May would argue that, judging by the name, Pavlova must be a female; I will also add that Pavlova is a cat.



The sequence of premises followed by a conclusion is called *argument*. Arguments are formally written in the following form, using the character ⊢ (called “turnstile”):

Premise_1, Premise_2, …, Premise_n ⊢ Consequence

Valid and Sound Arguments

As you see from examples above, some conclusions make sense, some don’t; also some premises make sense, and some don’t. Let’s disambiguate these situations.

Definition
An argument is called *valid* if the conclusion is true, assuming that the premises are true. An argument is called *sound* if it is valid, and the premises are true.

Note that we have just introduced the word “true” in our discourse. This is a little bit unusual; but we are not quite formal here.

You can check by yourself whether each of the examples above is valid or sound or neither or both.

Premises are called *inconsistent* if they contradict each other, that is, you can deduce $\bot$ (that is, *false*) out of them. Remember, in this case the argument is not sound (wrong premises!), but it is always valid. This may be the easiest way to prove literally anything – just start with inconsistent premises.

Proofs, Formally

Definition
A *proof* is a step-by-step demonstration that a conclusion follows from the premises (that is, that

the argument is valid).

How do we know that our proof makes sense? There are a variety of approaches. One of these is applying *rules*; we will walk through such rules, without questioning them (questioning them means proving theorems, and it is beyond the scope of this book).

Below are the proof rules; some of them are obvious, some less obvious.

Elimination Rule

Also known as *Indiscernibility of Identicals*, *Substitution Principle*, and *Identity Elimination*.

The rule is this:

P(a), a=b ⊢ P(b)

For example,
$x^2 > x^2 - 1, x^2 - 1 = (x+1)*(x-1) \vdash x^2 > (x+1)*(x-1)$

Introduction Rule

Also known as *Reflexivity of Identity*.

The rule is this:

P ⊢ x=x

Have you noticed that we are using variables already? Strictly speaking, we write this rule, but we mean anything can be substituting x.

From these two rules we can deduce symmetry and transitivity of identity:
a=b, a=a ⊢ b=a
a=b, b=c ⊢ a=c

Neat, right? Now let’s have more rules.

Negation Elimination

¬¬P ⊢ P

Informally, it means that if we have “non not P”, we can say that P holds. This may sound obvious, but look at it like this. We could not prove P. We only could prove that assuming “not P” leads to contradiction. Does it give us P? Probably not, generally speaking. But in this classical logic we assume that it does.

Conjunction Elimination

$P \land Q \vdash Q$

Informally, if we have a conjunction of P and Q, then we have Q. This is a part of definition of conjunction.

Conjunction Introduction

$P, Q \vdash P \land Q$

Informally, if we have P and Q, then we have a conjunction, $P \land Q$. This is a part of definition of conjunction.

Disjunction Introduction

$P \vdash P \lor Q$

Informally, if we have P, a disjunction of P and Q, also holds. This is a part of definition of disjunction.

Disjunction Elimination

$$\frac{P \lor Q, \; P \vdash R, \; Q \vdash R}{R}$$

The meaning of this schema above is the following: we have $P \lor Q$, and we have *subproofs* that P yields R and Q yields R; then we have R.

Negation Introduction

$$\frac{P \vdash \bot}{\neg P}$$

This can be considered as a definition of negation.

⊥ Introduction

$P, \neg P \vdash \bot$

This property of negation consists of deducing “bottom” from both P and its negation.

⊥ Elimination

$\bot \vdash P$

This is the property of “bottom”: anything follows from it. Good for proving existence of supernatural creatures or entities, as well as their non-existence.

Proof by Contradiction

The trick consists of the following: To prove ¬S, assume S, and deduce ⊥ (that is, *false*).

Example

Let's prove that sqrt(2) is irrational.

Assume it is rational, sqrt(2)=p/q , where p and q are natural numbers, mutually prime (have no common divisors). You may ask why mutually prime? Because common divisors don't count, p*x/q*x=p/q.

If we have sqrt(2)=p/q , then $p^2=2*q^2$.
Can p be odd? No! It is 2*something. But if p is even, $p^2={p_1}^2*4$, right? So $q^2=2*{p_1}^2$. So q is even too, so p and q do have a common divisor, oops. Contradiction! Our assumption was wrong.

Proof by Cases

To prove that P∨Q ⊢ R, it is enough to prove that P ⊢ R and Q ⊢ R.
This can be extended to a list of P1,P2,…,Pn and proving that P1∨P2∨…∨Pn ⊢ R

Example

Prove that a rational number a can be represented as b^c where both b and c are irrational numbers.
Proof. We know that sqrt(2) is irrational; now take $d=\mathrm{sqrt}(2)^{\mathrm{sqrt}(2)}$. If d is rational, we have it; if it is not, take $(d^{\mathrm{sqrt}(2)})^{\mathrm{sqrt}(2)}= d^{\mathrm{sqrt}(2)*\,\mathrm{sqrt}(2)} = \mathrm{sqrt}(2)^2 = 2$, and it is rational.

Conclusion

This was the first part of three; next we will see how we can do without Booleanness… and what does being Boolean means; then we cover quantifiers.