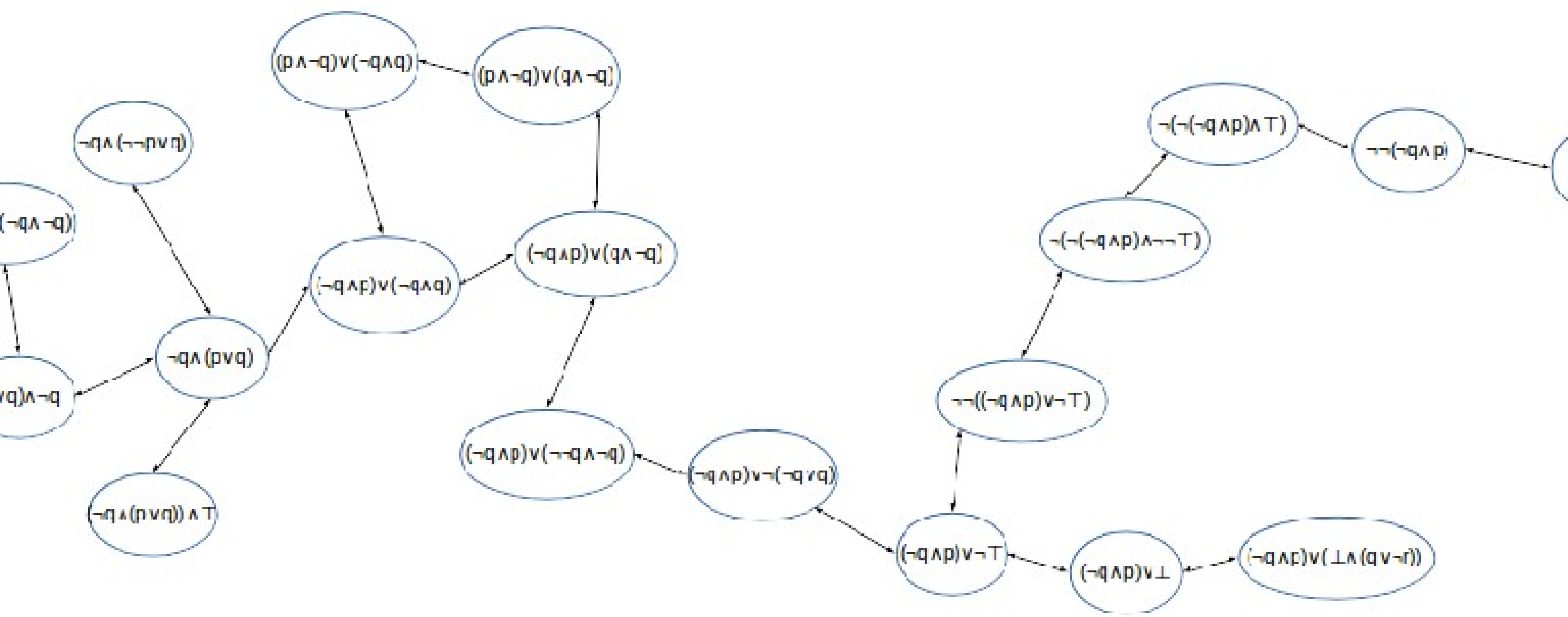


(p∧¬q)∨(¬q∧q)
(p∧¬q)∨(q∧¬q)
¬q∧(¬¬p∨q)
¬q∧(p∨q)
(¬q∧p)∨(¬q∧q)
(¬q∧p)∨(q∧¬q)
(¬q∧(p∨q))∧⊤
(¬q∧p)∨(¬¬q∧¬q)
(¬q∧p)∨¬(¬q∨q)
(¬q∧p)∨¬⊤
(¬q∧p)∨⊥
(¬q∧p)∨(⊥∧(q∨¬r))
¬¬((¬q∧p)∨¬⊤)
¬(¬(¬q∧p)∧¬¬⊤)
¬(¬(¬q∧p)∧⊤)

Logical Counterfactuals and Proposition graphs

Logical Counterfactuals and Proposition graphs, Part 1

Crossposted from the AI Alignment Forum. May contain more technical jargon than usual.

Within this sequence of posts I will outline a procedure for logical counterfactuals based on something similar to proof length. In this post I present a reimagining of propositional logic in which proving a theorem is taking a walk around a graph of equivalent propositiitions.

Within this post, I will use Latin symbols, for specific symbols within proof strings. I will sometimes use symbols like ∀ to represent a function with particular properties.

I will use Greek letters to represent an arbitrary symbol, upper case for single symbols, lower case for strings.

Respecifying Propositional logic

The goal of this first section is to reformulate first order logic in a way that makes logical counterfactuals easier. Lets start with propositional logic.

We have a set of primitive propositions $p, q, r, \ldots$ as well as the symbols $\top, \bot$. We also have the symbols $\vee, \wedge$ which are technically functions from $\text{Bool}^2 \to \text{Bool}$ but will be written $p \vee q$ not $\vee(p, q)$. There is also $\neg : \text{Bool} \to \text{Bool}$

Consider the equivalence rules.

1. $\alpha \equiv \neg\neg\alpha$

2. $\alpha \wedge \beta \equiv \beta \wedge \alpha$

3. $(\alpha \wedge \beta) \wedge \gamma \equiv \alpha \wedge (\beta \wedge \gamma)$

4. $\neg\alpha \wedge \neg\beta \equiv \neg(\alpha \vee \beta)$

5. $\alpha \wedge \top \equiv \alpha$

6. $\neg\alpha \vee \alpha \equiv \top$

7. $\bot \equiv \neg\top$

8. $\alpha \wedge (\beta \vee \gamma) \equiv (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$

9. $\bot \wedge \alpha \equiv \bot$

10. $\alpha \equiv \alpha \wedge \alpha$

Theorem

Any tautology provable in propositional logic can be created by starting at $\top$ and repeatedly applying equivalence rules.

Proof

First consider $\alpha \implies \beta$ to be shorthand for $\neg\alpha \vee \beta$.

Lemma

We can convert $\top$ into any of the 3 axioms.

$\alpha \implies (\beta \implies \alpha)$ is a shorthand for

$\neg\alpha \vee (\neg\beta \vee \alpha) \equiv_1$

$\neg\neg(\neg\alpha \vee (\neg\beta \vee \alpha)) \equiv_4$

$\neg(\neg\neg\alpha \wedge \neg(\neg\beta \vee \alpha)) \equiv_4$

$\neg(\neg\neg\alpha \wedge (\neg\neg\beta \wedge \neg\alpha)) \equiv_1$

$\neg(\neg\neg\alpha \wedge (\beta \wedge \neg\alpha)) \equiv_2$

$\neg(\neg\neg\alpha \wedge (\neg\alpha \wedge \beta)) \equiv_3$

$\neg((\neg\neg\alpha \wedge \neg\alpha) \wedge \beta) \equiv_4$

$\neg(\neg(\neg\alpha \vee \alpha) \wedge \beta) \equiv_6$

$\neg(\neg\top \wedge \beta) \equiv_7$

$\neg(\bot \wedge \beta) \equiv_9$

$\neg\bot \equiv_7$

$\top$

Similarly

$(\alpha \implies (\beta \implies \gamma)) \implies ((\alpha \implies \beta) \implies (\alpha \implies \gamma))$

$(\neg\alpha \implies \neg\beta) \implies (\beta \implies \alpha)$

(if these can't be proved, add that they $\equiv \top$ as axioms)

End Lemma

Whenever you have $\alpha \wedge (\alpha \implies \beta)$, that is equiv to

$\alpha \wedge (\neg\alpha \vee \beta) \equiv_8$

$(\alpha \wedge \neg\alpha) \vee (\alpha \wedge \beta) \equiv_1$

$(\neg\neg\alpha \wedge \neg\alpha) \vee (\alpha \wedge \beta) \equiv_4$

$\neg(\neg\alpha \vee \alpha) \vee (\alpha \wedge \beta) \equiv_6$

$\neg\top \vee (\alpha \wedge \beta) \equiv_1$

$\neg\neg(\neg\top \vee (\alpha \wedge \beta)) \equiv_4$

$\neg(\neg\neg\top \wedge \neg(\alpha \wedge \beta)) \equiv_1$

$\neg(\top \wedge \neg(\alpha \wedge \beta)) \equiv_2$

$\neg(\neg(\alpha \wedge \beta) \wedge \top) \equiv_5$

$\neg\neg(\alpha \wedge \beta) \equiv_1$

$\alpha \wedge \beta$

This means that you can create and apply axioms. For any tautology, look at the proof of it in standard propositional logic. Call the statements in this proof $p_1, p_2, p_3 \ldots$

suppose we have already found a sequence of substitutions from $\top$ to $p_1 \wedge p_2 \ldots \wedge p_{i-1}$

Whenever p_i is a new axiom, use (5.) to get $p_1 \wedge p_2 \ldots \wedge p_{i-1} \wedge \top$, then convert $\top$ into the instance of the axiom you want. (substitute alpha and beta with arbitrary props in above proof schema)

Using substitution rules (2.) and (3.) you can rearrange the terms representing lines in the proof and ignore their bracketing.

Whenever p_i is produced by modus ponus from the previous p_j and $p_k = p_j \implies p_i$ then duplicate p_k with rule (10.), move one copy next to p_j and use the previous procedure to turn $p_j \wedge (p_j \implies p_i)$ into $p_j \wedge p_i$. Then move p_i to end.

Once you reach the end of the proof, duplicate the result and unwind all the working back to $\top$, which can be removed by rule (5.)

Corollary

$\{p, q, r\} \vdash s$ then $p \wedge q \wedge r \equiv p \wedge q \wedge r \wedge s$

Because $p \wedge q \wedge r \implies s$ is a tautology and can be applied to get s.

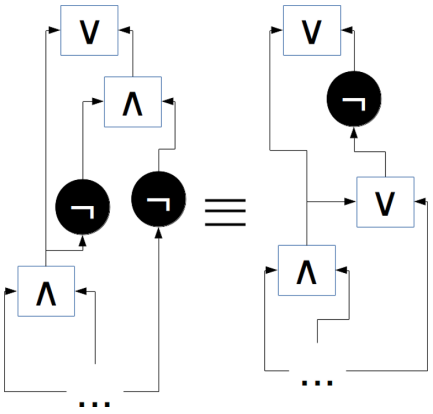
Corollary

Any contradiction is reachable from $\bot$

The negation of any contradiction k is a tautology.

$\bot \equiv \neg\top \equiv \neg\neg k \equiv k$

Intuitive overview perspective 1



An illustration of rule 4. ¬α ∧ ¬β ≡ ¬(α ∨ β) in action.

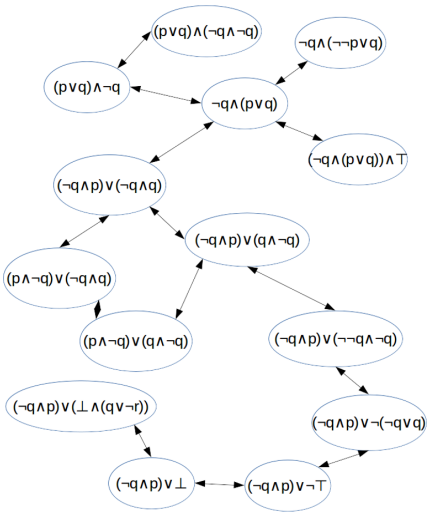
We can consider a proposition to be a tree with a root. The nodes are labeled with symbols. The axiomatic equivalences become local modifications to the tree structure, which are also capable of duplicating and merging identical subtrees by (10.). Arbitrary subtrees can be created or deleted by (5.).

We can merge nodes with identical subtrees into a single node. This produces a directed acyclic graph, as shown above. Under this interpretation, all we have to do is test node identity.

Intuitive overview perspective 2

Consider each possible expression to be a single node within an infinite graph.

Each axiomatic equivalence above describes an infinite set of edges. To get a single edge, substitute the generic $\alpha, \beta \ldots$ with a particular expression. For example, if you take (2. $\alpha \wedge \beta \equiv \beta \wedge \alpha$) and substitute $\alpha := p \vee q$ and $\beta := \neg q$. We find a link between the node $(p \vee q) \wedge \neg q$ and $\neg q \wedge (p \vee q)$.



Here is a connected subsection of the graph. Note that, unlike the previous graph, this one is cyclic and edges are not directed.

All statements that are provably equivalent in propositional logic will be within the same connected component of the graph. All statements that can't be proved equivalent are in different components, with no path between them.

Finding a mathematical proof becomes an exercise in navigating an infinite maze.

In the next Post

We will see how to extend the equivalence based proof system to an arbitrary first order theory. We will see what the connectedness does then. We might even get on to infinite dimensional vector spaces and why any of this relates to logical counterfactual.

Logical Counterfactuals and Proposition graphs, Part 2

Crossposted from the AI Alignment Forum. May contain more technical jargon than usual.

If you haven't read part 1, read it here.

In this post we extend the notion of propositional graphs to cover any first order theory.

Consider some arbitrary first order theory. For example a theory of integers.

We can consider the program to deal with two basic types. $B = \{\top, \bot\}$ and

$N = \{0, 1, 2, \dots\}$. All the functions and transformation rules in the previous post apply

to B. For example, $\top$ is of type B. $\wedge$ is a function from $B^2 \to B$.

3 and 7 are of type N, and $<$ is a function from $N^2 \to B$. So $3 < 7$ is a valid Boolean that can be treated just like p with respect to the propositional transformations. ie

$(3 < 7) \wedge p \equiv p \wedge (3 < 7)$.

We can also add $S : N \to N$ the successor function. Here are some more equivalence rules you might want.

$S\alpha < S\beta \equiv \alpha < \beta$

$\alpha < 0 \equiv \bot$

$0 < S\alpha \equiv \top$

$\neg(\alpha < \beta) \equiv \beta < S\alpha$

In general, there is a finite list of types. Call them $X_1, X_2, \dots X_n$

All functions have a fixed number of inputs, each with a fixed type, and output something of a fixed type. In our parse tree from before, this means that our edges have one of finitely many different colours, and that each symbol node has a fixed pattern of colors for its inputs and outputs. + is a node of type N and both its children

are of type N. $<$ is a node of type B, and both its children are of type N. $\neg$ is of type B

and has one child of type B. Where the output type is not clear, it may be indicated by a subscript. $<_B (3_N, 5_N)$

Before we can get to predicate logic, we will introduce a set of implicit variable symbols V_X for each type X in use. $V = \cup_{x=1}^{n} V_X$ is the set of all implicit variables. An implicit variable is an abstract symbol like $\perp$ or 3. Specific implicit variables will be represented by capital roman letters (ABC). Substitution rules can also contain references to arbitrary members of some V_X. Eg

$$Sub(\Gamma_N, \delta_N, +_N(\alpha_N, \beta_N)) \equiv +_N(Sub(\Gamma_N, \delta_N, \alpha_N), Sub(\Gamma_N, \delta_N, \beta_N))$$

$$Sub(\Gamma_N, \delta_N, \Gamma_N) \equiv \delta_N$$

Here Sub is the substitution function, the $\alpha_N, \beta_N, \delta_N$ represent any expression that has a real type. The Γ_N represents any single symbol from the set V_N.

Applying these equivalences tells us that

$$Sub(X, 5, +(3, X)) \equiv +(Sub(X, 5, 3), Sub(X, 5, X)) \equiv +(Sub(X, 5, 3), 5)$$

If 3 is a shorthand for S(S(S(0))) then the rules $Sub(\Gamma_N, \delta_N, S(\alpha_N)) \equiv S(Sub(\Gamma_N, \delta_N, \alpha_N))$ and $Sub(\Gamma_N, \delta_N, 0_N) \equiv 0_N$ allow deduction all the way to $+(Sub(X, 5, 3), 5) \equiv 8$

A general first order theory makes use of = and $\forall$ as well as [/] for substitution. These do not need any special properties, and can be defined by the normal substitution rules. Technically there is a collection of Sub's, one for each pair of types.

We have 4 axioms, which can be turned into rules by making them all equivalent to $\top$.

$$\forall_B(\Gamma_N, =_B, (\Gamma_N, \Gamma_N)) \equiv \top$$

$$\forall_B(\Gamma_N, \forall(\Delta_N, \implies_B(=_B (\alpha_N, \beta_N), =_B (\phi_B, Sub(\Gamma_N, \Delta_N, \phi_B))))) \equiv \top$$

$$\implies_B(\forall_B(\Gamma_N, \phi_B), Sub(\Gamma_N, \alpha_N, \phi_B)) \equiv \top$$

$$\implies_B(\forall_N(\Gamma_N, \implies_B(\phi_B, \psi_B)), \implies_B(\forall_N(\Gamma_N, \phi_B), \forall_N(\Gamma_N, \psi_B)))$$

And a rule that allows generalization.

$\forall_N(\Gamma_N, \top) \equiv \top$

I believe that this set of substitution rules, or something like it, can produce predicate logic. I also believe that more substitution rules and symbols can be added to make any first order theory. The intuition is that you can take any proof in any first order logical theory and convert it into a series of substitution rules by anding in any axioms, and applying moduls ponens. Generalization can be done by turning $\top$ into $\forall(X, \top)$ and then using substitutions not dependent on X to turn it into $\forall(X, p)$.

Theorem

For any set of symbols with appropriate type signatures, for any recursively enumerable equivalence relation H. (With the property that $a \equiv b \implies f(a) \equiv f(b)$)There exists a finite set of extra symbols and a finite set of substitution rules such that these equivalences hold.

Proof Outline

Introduce the symbols 0,1,end so that binary strings can be stored 0(1(1(0(end))))

Give them some new type Y, and add a load of type converters, symbols that take in an object of type Y, and output some other type.

Introduce + for concatenation.

$+(0(\alpha_Y), \beta_Y) \equiv 0(+(\alpha_Y, \beta_Y))$

And the same for 1

$+(\text{end}, \alpha_Y) \equiv \alpha_Y$

By adding the rules

$P(\alpha, 0(\beta)) \equiv 0(P(0(\alpha), \beta))$

$P(\alpha, 1(\beta)) \equiv 0(P(1(\alpha), \beta))$

$P(\alpha, \text{end}) \equiv 1(\alpha)$

Then if $|\alpha| = n$ then $P(\text{end}, \alpha) \equiv 0^n 1$ reverse(α).

This means that $J(\alpha, \beta) \equiv +(P(\text{end}, \alpha), P(\text{end}, \beta))$ can unambiguously express pairs of bitstrings as a single bitstring.

Arbitrary syntax trees can be encoded like
$*(\text{Conv}_N(\alpha_Y), \text{Conv}_N(\beta_Y)) \equiv \text{Conv}_N(J(010\text{end}, J(\alpha_Y, \beta_Y)))$

Where 010 is the unique bitstring representing "*", an arbitrary symbol.

Then add a rule that

$\text{Conv}_N(\alpha_Y) \equiv \text{right}_N(TM1(\text{left}(J(\alpha_Y, \beta_Y))))$

As β_Y can be anything, it could be an encoding of what you want. Then we let left(0(α)) ≡ 0(left(α)) and the same for one, to run left to the far end of the bitstring.

Then left(end) ≡ n(left(end)) where n is a null symbol for tape that doesn't store data and we can extend tape as needed. Using rules like TM1(n(α)) ≡ 1(TM3(α)) we can run any turing machine with one symbol per state. Then finally we have a rule like TM4(0(α)) ≡ success(TM4(0(α))). Let success commute with 0,1 so it can go up to the top. Let right(success(α)) ≡ right(success(n(α))).

Then you can unwind all the computations, and $\text{right}_N(\text{success}(TM1(\text{left}(J(\alpha_Y, \beta_Y))))) \equiv \text{Conv}_N(\beta_y)$ means that you can take the bit stream that you computed to be equal, and decode it back into its symbols.

Lemma

Any possible computably checkable formal proof system can be converted into an equivalence graph.

Post Ending.

Next time we will see when different proof systems are provably the same. I know the formatting is rough and the explanations aren't always as clear as could be. Tell me about any unusually opaque parts. If these ideas turn out to be important, someone will need to rework this. I know the axioms used are not the simplest possible, and proofs are not fully formalized.

Logical Counterfactuals and Proposition graphs, Part 3

Crossposted from the AI Alignment Forum. May contain more technical jargon than usual.

Notation note, [ ] are for lists, ( ) are for functions.

Note that many of the words and symbols I am using are made up. When this maths is better understood, someone should reinvent the notation. My notation isn't that great, but its hard to make good notation when you are still working out what you want to describe.

A theory (of my new type, not standard maths) T is formally defined to be,

$T = [\psi, \rho, \Xi, \text{type}, \text{arity}]$

Where $\psi = \{s_1, s_2, \dots\}$ are the theories symbols. These can be arbitrary mathematical objects.

$\Xi = \{\Xi_1, \Xi_2, \dots\}$ is the set of types, also arbitrary.

$\text{type} : \psi \to \Xi$ is a function.

$$\text{arity} : \psi \to \bigcup_{i=0}^{\infty} \underbrace{\Xi \times \Xi \times \cdots \times \Xi}_{i}$$

Is also a function.

An expression E in a theory is defined recursively, it consists of a pair

$E = [s, v_1, v_2, \cdots, v_n]$. Here $s \in \psi$ and $\forall 1 \leq i \leq n : v_i$ is an expression.

Let $\text{arity}(s) = [x_1, x_2, \dots x_m]$

Sometimes we will write type(E), what we really mean is type(s), and

arity(E) = arity(s) We write symbol(E) = s when we want to refer to just s and ignore

$v_1, \dots, v_n$

Expressions have the restriction that m = n the number of elements of Ξ that s is mapped to is the same as the number of other expressions.

We also insist that for all i, type(v_i) = x_i

The base case happens when arity(s) = [ ] the empty list.

All expressions are strictly finite mathematical objects. There are no self referential expressions.

Expressions can be written $s(v_1, \ldots, v_n)$

We can define equality between expressions $e = s(v_1, \ldots)$ and $f = t(w_1, \ldots)$ recursively by saying e = f iff symbol(e) = symbol(f) and forall i : $v_i = w_i$

The distinction between expressions e, f is defined recursively.

$e - f = [e, f]$ if symbol(e) $\neq$ symbol(f)

$e - f = [e, f]$ if $\exists i \neq j \in N : v_i \neq w_i$ and $v_j \neq w_j$

$e - f =$ None if e = f

$e - f = v_i - w_i$ if $\exists_1$ i : $v_i \neq w_i$

These can be uniquely expressed as strings made up of $\Xi \cup \{ ´(´, ´,´, ´)´ \}$

Lets define $V(n) = V(\Xi_n)$ to be the set of all possible expressions of type Ξ_n.

A function $f : V(n_1) \times \ldots V(n_t) \to V(m)$ is called **basic** if it is constant, or it is the projection function (so $f(e_1, \ldots e_t) = e_k$ for fixed $k \leq t$) or f can be expressed as

$f(e_1, \ldots e_t) = s(v_1, \ldots v_n)$ where s is constant and each v_i is a **basic** function of $e_1, \ldots e_t$.

Note you can cross as much extra junk into f as you like and ignore it. If $f(e_1, e_2)$ is basic, so is $g(e_1, e_3, e_2, e_4) = f(e_1, e_2)$.

Basic functions can be said to have type(f) = Ξ_m and arity(f) = $[\Xi_{n_1}, \ldots \Xi_{n_t}]$

Basic functions can be defined in the style of $f(\alpha, \beta) = s_1(c_1, \alpha, s_2(\alpha, \beta))$

Finally, $\rho = \{[f_1, g_1], [f_2, g_2], \cdots\}$ where f_i and g_i are basic functions with the same domains and codomains. type(v_i) = type(w_i)

We write $x(\alpha) \equiv y(z(\alpha))$ to mean that the pair of basic functions $f(\alpha) = x(\alpha)$ and $g(\alpha) = y(z(\alpha))$ are matched in ρ. Ie $[f, g] \in \rho$ where $x, y, z \in \psi$

We express the concept that for expressions e, f that $e - f = [p, q]$ and there exists $[f, g] \in \rho$ and expressions $v_1, \ldots v_n$ such that $p = f(v_1, \ldots v_n)$ and $q = g(v_1, \ldots v_n)$ by saying

$e \equiv f$.

We express that $\exists\ e_1, e_2, \ldots, e_n$ such that $\forall\ i < n : e_i \equiv e_{i+1}$ and $e_1 = e$ and $e_n = f$ as $e \sim f$. (previous posts use $\equiv$ for both concepts)

Lets create a new theory, called T1, this is a very simple theory, it has only 1 constant, 2 functions and 2 substitution rules, with only a single type.

$a(b(\alpha)) \equiv \alpha$

$b(a(\alpha)) \equiv \alpha$

With the only constant being 0. This theory can be considered to be a weak theory of integers with only a +1 and -1 function. It has a connected component of its graph for each integer. Propositions are in the same graph if and only if they have the same count(a) − count(b). Theorems look like $a(a(a(b(b(0))))) \sim b(a(a(0)))$.

Now consider the theory T2 formed by the symbols f, g, h

$f(f(f(g(g(\alpha))))) \equiv \alpha$

$f(\alpha) \equiv h(g(\alpha))$

$f(h(\alpha)) \equiv h(f(\alpha))$

$g(h(\alpha)) \equiv h(g(\alpha))$

It turns out that this theory also has one connected component for each integer, but this time, propositions are in the same component if
$2 \times \text{count}(f) - 3 \times \text{count}(g) + 5 \times \text{count}(h)$ is the same.

When $S = (\psi_S, \rho_S, \Xi_S, \text{type}_S, \text{arity}_S)$ is a theory and similarly for T.

We can define the disjoint union, $S \sqcup T$ to be the theory

$(\psi_S \sqcup \psi_T, \rho_S \sqcup \rho_T, \Xi_S \sqcup \Xi_t, \text{type}_S \sqcup \text{type}_T, \text{arity}_S \sqcup \text{arity}_T)$

Consider a function $f : \Xi_S \to \Xi_T$ and a function $Q : \psi_S \to$basic functions in T, such that

$\text{type}(Q(s_i)) = f(\text{type}(s_i))$ and where$\text{arity}(s_i) = [X_1, \ldots X_n]$ means

$\text{arity}(Q(s_i)) = [f(Q(s_1)), \ldots f(Q(s_n))]$.

These arity conditions mean that for any expression $e = s(v_1, \ldots v_n)$ i n S, we can define an expression in T

V_T is the set of expressions in T. Call $Q^* : V_S \to V_T$ a transjection if it meets the condition that $Q^*(e) = Q(s)(Q^*(v_1), \ldots Q^*(v_n))$. For each Q meeting the above conditions, there exists exactly one transjection.

We can now define a relation.

We say $S \lesssim T$ iff there exists $Q^* : V_S \to V_T$ a transjection such that $e \sim f$ in S iff $Q^*(e) \sim Q^*(f)$ in T. Call such transjections projections.

Say $S \approx T$ iff $S \lesssim T$ and $T \lesssim S$.

Theorem

$S \lesssim T$ and $T \lesssim U$ implies $S \lesssim U$.

Proof

There exists a $Q : V_S \to V_T$ and $R : V_T \to V_U$ projections.

The composition of basic maps is basic, by the recursive definition.

The composition of transjections is a transjection.

So $R \circ Q$ is a transjection.

For all $e, f \in V_S$, then $e \sim f \iff Q(e) \sim Q(f) \iff R(Q(e)) \sim R(Q(f))$

Hence $R \circ Q$ is a projection.

Lemma

$S \lesssim S$

Let Q be the identity. The identity map is a projection.

Theorem

Suppose S and T are theories, with $Q : V_S \to V_T$ and $R : V_T \to V_S$ transjections.

Suppose that for all $e = s(v_1, \ldots, v_n) \in V_S$ we know that $R(Q(e)) \sim e$.

And the same when we swap S with T. Call Q and R psudoinverses.

If we also know that for all $[f(v_1, \ldots v_n), g(v_1, \ldots v_n)] \in \rho_S$ a pair of basic functions, that for all $v_1, \ldots v_n$ we know $Q(f(v_1, \ldots)) \sim Q(g(v_1, \ldots))$. Say that Q is axiom preserving.

Again we know the same when S is swapped with T, IE that R is axiom preserving.

Note that all these claims are potentially provable with a finite list of $a \sim b \sim c \ldots$ when the expressions contain the arbitrary constants $v_1, \ldots v_n$.

All the stuff above implies that $S \approx T$.

Proof

Consider $e \equiv f \in V_S$. We know that $e - f = [a, b]$. st there exists $[f, g] \in \rho_S$ and some $v_1, \ldots$ such that $f(v_1, \ldots) = a$ and $g(v_1, \ldots) = b$ (or visa versa.)

This tells us that $Q(a) \sim Q(b)$

If $\forall i : Q(v_i) \sim Q(w_i)$ then $Q(s(v_1, \ldots v_n)) \sim Q(s(w_1, \ldots w_n))$. True because Q(s) is a basic function, and they preserve similarity.

Repeat this to find that $Q(e) \sim Q(f)$.

If $e \sim f$ then $e = e_1$, $f = e_n$ and $e_i \equiv e_{i+1}$, so $Q(e_i) \sim Q(e_{i+1})$ so $Q(e) \sim Q(f)$.

On the other hand, if $Q(e) \sim Q(f)$ then $R(Q(e)) \sim R(Q(f))$ because the same reasoning also applies to R. For any $e', f' \in V_T$ we know $e' \sim f' \implies R(e') \sim R(f')$.

We know $Q(e), Q(f) \in V_T$. But S and T are psudoinverses, so $e \sim R(Q(e)) \sim R(Q(f)) \sim f$ hence $e \sim f \iff Q(e) \sim Q(f)$

Therefore $S \lesssim T$. Symmetrically, $T \lesssim S$ so $S \approx T$

Remember our theories T1 and T2 from earlier? The ones about a,b and f,g,h?

We can now express the concept that they are basically the same. $T1 \approx T2$.

We can prove this by giving basic functions for each symbol, that generates transjections, and by showing that they are psudoinverses and axiom preserving, we know they are projections and $T1 \approx T2$.

$Q : T1 \to T2$

$Q(0) = 0$, $Q(a(\alpha)) = f(f(g(Q(\alpha))))$, $Q(b(\alpha)) = f(g(Q(\alpha)))$.

$R : T2 \to T1$

R(f(α)) = a(a(R(α))) , R(g(α)) = b(b(b(R(α)))) , R(h(α)) = a(a(a(a(a(R(α)))))) , R(0) = 0 .

For example, pick the symbol a. To show that Q and R are psudoinverses, we need to show that R(Q(a(α))) ~ a(α). We know

R(Q(a(α))) = R(f(f(g(Q(α))))) = a(a(a(a(b(b(b(R(Q(α))))))))) ~ a(a(a(a(b(b(b(α))))))) ~ a(α) .

To prove these transjections to be psudoinverses, do this with all symbols in $\psi_S \sqcup \psi_T$.

Finally we prove that Q is axiom preserving. We must show that Q(a(b(α))) ~ Q(α) and that Q(b(a(α))) ~ Q(α) .

Q(a(b(α))) = f(f(g(f(g(Q(α)))))) ≡ f(f(g(h(g(g(Q(α))))))) ≡ f(f(h(g(g(g(Q(α))))))) ≡ f(f(f(g(g(Q(α)))))) ≡ Q(α)

Likewise Q(a(b(α))) ~ Q(α) .

R is also axiom preserving.

So T1 ≈ T2.

Conclusion

We have formally defined a notion of a theory, and provided a criteria for telling when two theories are trivial distortions of each other. This will allow us notions of logical provability that aren't dependent on an arbitrary choice of formalism. By looking at all equivalent theories, weighted by simplicity, we can hope for a less arbitrary system of logical counterfactuals based on something thematically similar to proof length, although kind of more continuous with graduations of partly true.