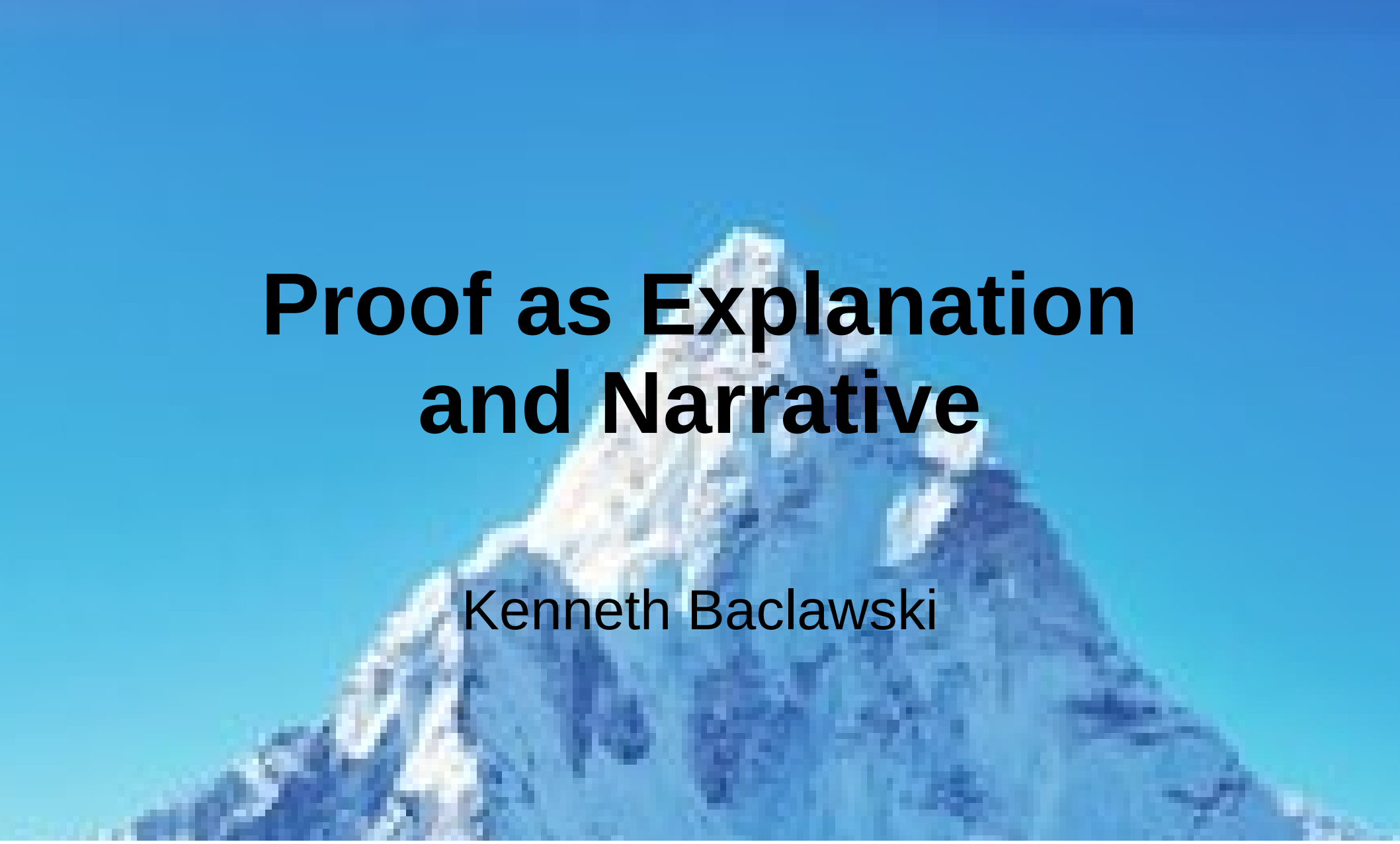


Proof as Explanation and Narrative

Kenneth Baclawski

Ontology Summit 2019

Background I

- “Gold Standard” concepts of explanation:
  - Proof – deductive, in a formal knowledge representation
  - Causal relationships
- Other concepts and aspects of explanation
  - Provenance, transparency, dialogue, …
- However, there are many issues
  - Confusion about concepts, ignorance about what is practical, mission creep, etc.
  - What *is* a proof and *how* is it an explanation?
- Source: Benjamin Grosof Presentation

Background II

- Challenge: Design and *evaluate* a software system that:
  - Represents commonsense knowledge
  - Supports reasoning (such as deduction and explanation)
- Solution for everyday tasks: Physical Turing Test
- Strategy: Build larger theories from components
- However there are many issues:
  - Logical reasoning/proof is fundamental, but now in a theory built from components.
  - How can one prove and explain in such a framework?
- Source: Michael Grüninger Presentation

Outline

- Example of a proof
- Introduction to a framework for proofs
- Dealing with complexity
- Explanation support
- More advanced proofs
- Pay close attention: There will be quiz.

Example of a Proof

Example of a Syllogism

- Consider this classical syllogism:
  - All men are mortal.
  - Socrates is a man.
  - Therefore, Socrates is mortal.
- We will examine two approaches for proving that the conclusion is correct.
  - Proof as explanation: One finds a classical proof which one then presumes is sufficient as an explanation.
  - Explanation as proof: The focus is on explanation from the start, and a proof arises as a consequence of the explanation.

Proof as Explanation

- The classical syllogism might be rendered like this:
  - Assume: (forall (x)(implies (Man x) (Mortal x)))
  - Assume: (Man Socrates)
  - Conclude: (Mortal Socrates)
- How is the conclusion actually proved?
  - Some kind of application of the first statement
  - A theorem prover was applied and it verified the conclusion
  - Some rules were applied
  - Some informal reasoning was used
  - Some other approach

Shortcomings of Proof as Explanation

- Proofs have the *potential* to be explanations but they have shortcomings
  - Most proofs in publications are only informal sketches
    - The reader is expected to fill in the details
    - Some published proofs later turn out to be wrong
    - May still be rather poor narratives
  - A theorem prover is better but it may be a “black box”
    - The details may not be shown at all
    - If details are shown, they are in an obscure notation and cannot easily be independently verified.

Explanation as Proof

- A new approach: Explanation *is* the proof
  - For a statement S, let Ex(S) be the collection of explanations for why one should accept S.
  - The goal is no longer to show that S is true or false but to find an explanation for S.
- This approach goes by various names
  - Propositions as types
  - Types as terms
- Developed independently by many individuals: Kolomogorov (1932), Heyting (1934), Curry and Feys (1958), de Bruijn (1967)...

Introduction to a framework for proofs based on explanation

Introduction to Explanation Logic

- A traditional logic must provide:
  - Language for constructing statements
  - Ability to assert statements as axioms and to specify rules
  - Mechanism for determining whether a statement is true or false
- We will replace the last requirement above by a mechanism for explanations.
- Disclaimer: Some technical details were omitted for pedagogical purposes.

Starting Point

- To avoid an infinite regress, one must start with at least one “built-in” entity from which others can be declared using axioms.
- This entity is the “collection” of collections, and is written **Collection**.
  - Written in boldface to emphasize that it is built-in.
  - **Collection** cannot itself be a collection because that would imply that it is an instance of itself.

Axioms in Explanation Logic

- What is an axiom or rule from this point of view?
  - An assertion that an entity has an instance
    - For example, asserting that Ex(S) has an instance means that it is being explained/accepted *a priori.*
  - One should also annotate the assertion with a citation or some rationale to answer the question “Why?”
- The notation e : C will mean that e is an instance of C.
  - For example, the collection of all statements can be defined by the axiom:

    Statement : **Collection**

Syllogism Example

- Return to the Aristotelian syllogism:
  - All men are mortal.
  - Socrates is a man.
  - Therefore, Socrates is mortal.
- The first axiom defines the universal collection of things:

  Thing : **Collection**
- This will declare Socrates to be a thing:

  Socrates : Thing
- However, we want to assert that Socrates to be a man.
  - Need a notion of class.
  - This is done with a function.

Functions

- A function (of one argument) consists of two parts:
  - The domain (i.e., the type of the argument)
  - The body which defines the value on each instance of the domain
- The notation for a function is:

  [x:Domain] Body or λ:Domain.Body

- For example, the function from Thing to Thing that maps each thing to the *same* thing is

  [x:Thing] x or λ:Thing.x

Function Application

- The application of a function Func to an argument arg is written by concatenating Func and arg.
  - Usually written (Func arg) but Func(arg) is also okay.
- Apply a function [x:Domain] Body to an argument arg by substituting arg for every occurrence of x in Body.
  - For example, (([x:Thing] x) arg) = arg
  - This is called *reduction*.
  - Two terms are *equivalent* if one can apply a series of reductions to each one so that one obtains the same term.

Collection of Functions

- Given collections C and D, the collection of all functions from C to D is written $C \rightarrow D$.
- Note that this notation is compatible with the mathematical notation. For example,

  $f: C \rightarrow D$

  - Mathematically, f is a function from C to D
  - In Explanation Logic, it means f is an instance of the collection $C \rightarrow D$

Expressing Classes

- In Classical Logic, one introduces one-place predicates (Man x) and (Mortal x) that determine whether x is a Man or is Mortal.
  - In general, a class is a one-place predicate
- In Explanation Logic, a class is a one-place function (C x), but now the value of (C x) is the collection of explanations for why x is in the class C.
- For the syllogism we have these three axioms:

  Man : Thing → **Collection**

  Mortal : Thing → **Collection**

  Socrates_is_man : (Man Socrates) // “Socrates is a man”

Implication

- We want to express the axiom that all men are mortal. In other words,
  - If a thing is a man then a thing is mortal.
- In classical logic $S \Rightarrow T$ is either true or false
- Explanation Logic requires an explanation for why S should imply T
  - There should be a function that maps each explanation of S to an explanation of T
  - The explanations of $S \Rightarrow T$ are $\{f: Ex(S) \rightarrow Ex(T)\}$
  - In other words, $\{f: (Ex\ S) \rightarrow (Ex\ T)\} = (Ex\ S) \rightarrow (Ex\ T)$

Implication for Socrates

- The fact that Socrates is a man implies Socrates is mortal could be expressed like this:

  Socrates_man_implies_mortal : (Man Socrates) → (Mortal Socrates)

- In other words, there is a function that maps any explanation of Socrates being a Man to an explanation of Socrates being a Mortal.
- However, this is not what we want. We want an axiom that asserts that *every* man is mortal.
  - More precisely, for every thing, if the thing is a man then the thing is mortal.
- For this we need a new kind of function, one whose body is a collection not an object.

Subsumption

- We can now express the axiom that Man is a subclass of Mortal:

  man_is_mortal : [x:Thing] (Man x) → (Mortal x)

- For any thing x, there is a function that maps any explanation of x being a Man to an explanation of x being a Mortal.

The Syllogism Revisited

- Our conjecture for the explanation for Socrates to be mortal is

  (man_is_mortal Socrates Socrates_is_man)

- To state this explanation in natural language one might say “The fact that all men are mortal in the case of Socrates can be applied to the fact that Socrates is a man to conclude that Socrates is mortal.”
- This sounds plausible, but how can one formally verify that our conjecture is correct?

The problem of verification

- To verify that our conjecture is correct we must show that

(man_is_mortal Socrates Socrates_is_man) : (Mortal Socrates)

- More generally, how can one find what a term is an instance of?
  - In other words, how can one find the type (or category) of a term?
- For a term T, let **category**(T) be the category/type of T.
  - Note that the function category is a meta-function, and is not one of the functions in the logic.

Verification

The meta-function **category** is computed recursively as follows:

1. For a variable x, bound by an axiom x : T or by [x:T], **category**(x) = T
2. **category**([x:Domain] Body) =

   [x:Domain] **category**(Body)
3. **category**(func arg) = (**category**(func) arg)

Verification that Socrates is Mortal

- Let's verify that our explanation is correct

  category(man_is_mortal Socrates Socrates_is_man)

  = (**category**(man_is_mortal) Socrates Socrates_is_man)

  = (([x:Thing] (Man x) → (Mortal x)) Socrates Socrates_is_man)

  = (((Man Socrates) → (Mortal Socrates)) Socrates_is_man)

  = (Mortal Socrates)

- Thus we have verified that

  (man_is_mortal Socrates Socrates_is_man) : (Mortal Socrates)

Dealing with complexity

Strategies for Dealing with Complexity

This section considers how to deal with complex explanations.

- Proof Assistance
- Patterns/Motifs
- Constructing from Components
- Narrative

Proof Assistance

Term Reconstruction

- One feature of proof checking systems is the ability to reconstruct some unspecified terms
  - In programming languages this feature is called type inference.
  - Proof checking systems are more general than programming languages because they can reconstruct more than just types.
  - This is not the same as theorem proving, which is much harder

Term Reconstruction Example

- For example, the explanation for Socrates being mortal could have been written (man_is_mortal ? ?) and the Explanation Logic system would fill in the missing terms to complete the proof.
- This example illustrates the concept of *unification*, which is one of the techniques used for term reconstruction.
- A good proof checker can fill in 80% or more of an explanation.

Patterns/Motifs

Patterns/Motifs

- A pattern or motif is a template for expressing a commonly occurring logical or rhetorical relationship.
- The Class motif is:

  Class : Individual → **Collection**

  - Declare a class

    instance_is_in_Class : (Class instance)

  - “The instance is a member of Class”

More Motifs

- The AND relation motif is:

$$\text{AandB} : A \rightarrow (B \rightarrow C)$$

  - “If A and B then C”
  - The parentheses are not needed since $\rightarrow$ is right-associative.
- The OR relation motif is

$$\text{AorB} : [G{:}\text{Goal}]\ (A \rightarrow G) \rightarrow (B \rightarrow G) \rightarrow G$$

  - “For every goal G, if A implies G and B implies G, then G holds”

Still More Motifs

- Universal quantification

  Every_x : [x:Index] A

  - “For every x in an index set, A holds.”
- Existential quantification

  Some_x : [G:Goal] ([x:Index] A → G) → G

  - “For every goal G, if for every x in an index set, A implies G, then G holds”

Mathematical Induction Motif

Nat : Collection

One : Nat

Succ : Nat → Nat

Induction : [f:Nat → Collection]

(f One)

→ ([n] (f n) → (f (succ n))

→ ([n] (f n)

- Note the use of term reconstruction above.

Constructing from Components

Building from Components

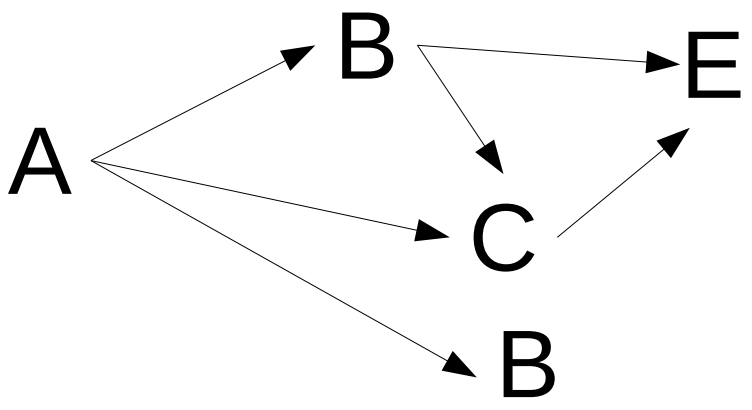
- Another approach to building and understanding explanations is to “divide and conquer”
  - Build large theories from small components.
- But as the number of components increases, the possibility of inconsistencies increases.
  - It also becomes difficult to manage.
- Automated construction can address these concerns.

Theory Morphisms

- A set of assertions (axioms and rules) is a *theory*.
- A mapping from one theory to another theory is a *theory morphism* if categories (types) are preserved.
  - Intuitively, a theory morphism preserves “truth”, or more precisely, preserves proofs.
- To check that a mapping is a theory morphism, one must verify that every assertion is mapped to another assertion or is a theorem.
  - One can automate the checking of a theory morphism.

Commutative Diagrams

- A collection D of theory morphisms is commutative if for any two theories S and T, the composition of all theory morphisms from S to T in D are the same.



  - The complexity of checking commutativity is very high, but I found an efficient algorithm for this purpose.
  - Note that one can have multiple copies of a theory.

Theory Combination

- The *combination* of a collection of theories related to each other by theory morphisms is defined by the colimit.
- The *colimit* is the (disjoint) union of the theories, such that some of the assertions are set equal to each other.
  - Theory morphisms specify the common parts of theories that are being combined.
  - An entity in the colimit can originate from entities in many components.
  - The provenance of an entity in a theory combination can be complicated, especially if the theories in a combination are themselves theory combinations.

Narrative

Narrative

- One can use linguistic methodologies to build human understandable narratives from formal proofs.
- One such methodology is Rhetorical Structure Theory (RST), but there are other methodologies.
- Combining Explanation Logic with RST is still in the early stages, but shows promise.
- Donna Fritzsche and Mark Underwood are presenting more about this subject.

Explanation Logic Support

The Explanation Logic System

- The acronym for Explanation Logic is XΛ
  - XΛ is similar to pure type systems, but it is not a pure type system.
  - Nevertheless, XΛ has the same properties as a pure type system.
- Baclawski's Modular Ontology System (BMOS) is my prototype system that supports XΛ
- Primary motivations:
  - Support for constructing ontologies from components
  - Ontology repository

Properties of XΛ

- Every term is a type (No lowest “object” level)
- Every term has a type (No highest level)
- Confluence (Church-Rosser Theorem)
- Predicativity
- Unique Typing
- Subject Reduction
- Higher-order logical framework
  - Other logics can be expressed in XΛ

Features of BMOS I

- Checks type consistency of terms
- Explanation/proof checking
- Tests equivalence of terms
- Term reconstruction
- Provenance annotations
- Theory modules
- Theory morphisms
  - Checked for preservation of proofs

Features of BMOS II

- Commutative diagrams
  - Checking of commutativity
- Colimit construction
  - Provenance is maintained
- Serialization in XML, JSON and human readable
- Web Service
  - Both SOAP and ReST at the same time
  - Ontology/Theory Repository

More advanced proofs

Implication Again

- Recall that we identified the explanations of S ⇒ T with the collection of functions (Ex S) → (Ex T)
- Unfortunately, we don't yet have a formal way of expressing what S ⇒ T is.
- We would like to say that S ⇒ T is a statement, but that seems to require that we have a function of two arguments

  ⇒ : Statement x Statement → Statement

- It looks we need a notion of Cartesian Product.

Currying

- The Cartesian Product can be avoided with a technique introduced by Frege (1893) but named after Curry
  - Applying the first argument produces a function of the remaining arguments
  - This requires that we allow the result of a function to be a function, i.e., functions must be first-class objects
- So the ⇒ function is now

  ⇒ : Statement → (Statement → Statement)
  - The parentheses can be omitted if we assume that → is right associative
  - To be consistent, we should write S ⇒ T like this (⇒ S T)

Statements and Explanations

- We can now make sense of (Ex S).
- It seems that one could simply assert something like this:

  Ex : Statement → Explanation

- However, this does not work because the value of (Ex S) is a collection of explanations not a single explanation.
- So as we did with classes like Man and Mortal the function Ex is defined by asserting

  Ex: Statement → **Collection**

- One can think of Ex as defining the (explanations for the) subset of statements that are true just as Man defines the subset of things that are men. If a statement has no explanations, then it is false.

Implication Continued

- Now that we have a meaning for S ⇒ T, how can we make sense of Ex(S ⇒ T) “=” (Ex S) → (Ex T)?
- They cannot actually be equal because all collections are disjoint!
- The trick is to axiomatically assert that there are functions in both directions.
  - The functions are used to construct new explanations
- We also need to state these axioms so that they apply to any two statements S and T.

Implication Axioms

- Here are the two implication axioms:

  IAX1 : [S][T] (Ex (⇒ S T)) → ((Ex S) → (Ex T))

  IAX2 : [S][T] ((Ex S) → (Ex T)) → (Ex (⇒ S T))

- Note the use of term reconstruction. One can write [S] rather than [S:Statement] because the missing term can easily be reconstructed.
  - In a programming language one would say that one has *inferred* the types of S and T.
- There is no requirement that these two functions be inverses of each other or have any other properties other than that they exist.

Example Explanation/Proof

- Let's try to explain/prove a more complicated theorem.
- We want to show that for any statement R, R ⇒ R.
- More precisely, we want to construct an instance of

  [R:Statement] (Ex (⇒ R R))

- Looking at the axioms, the only one that could possibly produce an instance of (Ex (⇒ R R)) is IAX2. Applying that axiom requires an instance of (Ex R) → (Ex R) .
  - This strategy is called *backward chaining*

Explanation/Proof Continued

- So far we know that the desired explanation has the form of the function (IAX2 R R) applied to an instance of

  (Ex R) → (Ex R).

- An instance of (Ex R) → (Ex R) is a function from (Ex R) to (Ex R).
- Any function will do, and the easiest one is the one that takes each explanation of R to the same explanation. In other words,

  [x:(Ex R)] x

- Putting these together, the explanation/proof of R ⇒ R is:

  (IAX2 R R ([x:(Ex R)] x))

Explanation/Proof Completed

- Finally, we want this for any statement R, so we construct a function of R:

  [R:Statement] (IAX2 R R ([x:(Ex R)] x))

- Assuming term reconstruction, it could have been written like this:

  [R](IAX2 ? ? ([x]x))

Verifying the Explanation

Verification that our explanation/proof is correct:

category([R:Statement] (IAX2 R R ([x:(Ex R)] x)))

= [R:Statement] **category**(IAX2 R R ([x:(Ex R)] x))

= [R:Statement] (**category**(IAX2) R R ([x:(Ex R)] x))

= [R:Statement] (([S:Statement][T:Statement] (((Ex S) → (Ex T)) → (Ex (⇒ S T))))

R R ([x:(Ex R)] x))

= [R:Statement] (([T:Statement] (((Ex R) → (Ex T)) → (Ex(⇒ R T)))) R ([x:(Ex R)] x))

= [R:Statement] ((((Ex R) → (Ex R)) → (Ex(⇒ R R)))) ([x:(Ex R)] x))

= [R:Statement] (Ex (⇒ R R))

Therefore the explanation has the desired category.

Quiz

The following conjectures are the first two of the three Łukasiewicz axioms for propositional logic. The third conjecture (modus ponens) is the only Łukasiewicz *rule* for propositional logic. The third Łukasiewicz axiom uses negation which has not yet been defined so it was omitted. Try to explain/prove these conjectures.

1. [P][Q] (Ex (⇒ P (⇒ Q P)))
2. [P][Q][R] (Ex (⇒ (⇒ P (⇒ Q R)) (⇒ (⇒ P Q) (⇒ P R)))
3. [P][Q] (Ex P) → (Ex (⇒ P Q)) → (Ex Q)

Acknowledgements

Michael Grüninger “Ontologies for the Physical Turing Test” http://bit.ly/2RiQSFz

Benjamin Grosof “An Overview of Explanation: Concepts, Uses, and Issues” http://bit.ly/2R9iD30

A. Appel. Hints on Proving Theorems in Twelf, February 2000. http://bit.ly/2AUB6dv