Sound Logic without Paradox

David A. Fisher[1,2,*], Stephen Fisher[3] and Donna M. Lambie[2]

[1]Carnegie Mellon University, Pittsburgh, PA
[2]Reasoning Technology LLC, Pittsburgh, PA
[3]ComputerWorks AG, Basel, Switzerland

Abstract

One of the greatest challenges for rule-based systems has been to invent a deductive logic that is sound for all possible domains and devoid of paradoxes. Without soundness, valid results cannot be guaranteed. The possibility of paradoxes make it impossible to prove soundness. The world today is dependent on logics that do not satisfy these criteria. This paper describes a logic of property-based types that emulates certain aspects of human deductive reasoning to create a logic that is sound, even for Gödel incomplete domains, and precludes paradoxes.

Keywords

Deductive logic, incomplete logic, paradox, property-based types, rule-based systems, consistency, soundness, human deductive reasoning

1. Background

Since, at least the time of Aristotle (384–322 B.C.E.) [1], there have been attempts to understand human deductive reasoning as a logical process for creating new knowledge that is entailed by what is already known. Knowledge expresses the theoretical and practical understanding of things in terms of their relationships to other things. A logic is a reasoning process that is conducted or assessed according to strict principles of validity.

Aristotle developed several Laws of Thought to characterize constraints necessary for sound reasoning:

- Law of Identity — for every individual X, X=X (i.e., what is is);
- Law of Noncontradiction — nothing can be both true and false
- Law of Excluded Middle — every statement is either true or false

Philosophers of the late 19th and early 20th centuries developed a formal deductive logic that is based on Aristotle's laws of though and is now called classical logic. Classical logic (CL) is the best known and most widely used logic today. It is the logic commonly taught in school. It serves as the foundation for much of mathematics and digital computing. It complements

RuleML+RR'23: 17th International Rule Challenge and 7th Doctoral Consortium, September 18–20, 2023, Oslo, Norway
*Corresponding author.
dafisher@ieee.org (D. A. Fisher); sfisher@computerworks.ch (S. Fisher); dlambie2000@yahoo.com (D. M. Lambie)
https://www.computer.org/profiles/david-a-fisher/ (D. A. Fisher)

heuristic and statistical inference methods used in artificial intelligence and machine learning applications.

A formal deductive logic is a rule-based symbol manipulation system for proving whether certain relationships are necessary consequences of known relationships. A deductive logic enables rigorous proof that conclusions are entailed by the premises. A *proposition* expresses knowledge in the form of a relationship among things. Propositions correspond to declarative sentences in natural language. A formal logic includes a language or symbolic notation to allow clear and unambiguous expression of propositions and other logical structures.

A formal deductive logic comprises a language, tautologies, a set of axioms, and a proof process. A *tautology* characterizes a relationship among propositions that is true for all propositions independent of an application. An *axiom* is a proposition that represents knowledge that is already known or assumed to be true about an application domain.

A *proof* is a process for generating new knowledge that is entailed by what is already known. A *theorem* is a proposition that represents new knowledge resulting from a proof. An *inference rule* is an interpretation of a tautology as a transformation or substitution rule that can be applied to a proposition to create a new proposition. An inference rule alters the form, but not the value, of a propositional expression. Each proof repeatedly applies the inference rules to the axioms and previous theorems to produce new theorems.

Consistency is of primary importance in a deductive logic. A set of propositions is consistent if and only if its members do not contradict each other. If the axioms are inconsistent, they cannot all be true, the logic is unsound, and the axioms describe a domain that cannot possibly exist. If the axioms are consistent, they describe some real or imaginary domain (but not necessarily the domain their author had in mind).

Soundness is the primary measure of success in a deductive logic. An inference rule is valid if it produces only true propositions when applied to true propositions. A logic is *valid* if and only if all of its tautologies are valid. A logic is *sound* if and only if its inference rules are valid and its axioms are true.

2. Challenge

Classical logic has at least two limitations that fall short of its original objectives. It suffers paradoxes and is unsound for certain domains. Intuitively, a paradox is something at odds with common sense. In formal logic, a paradox is an inconsistency that cannot be proven within its logic. The validity of any proof in a logic with paradoxes cannot be guaranteed. The discovery of Russell's paradox in 1901 [2] brought set theory into question as the foundation for mathematics. Because some paradoxes involve propositions that reference themselves (e.g., this sentence is false), proposals to eliminate paradoxes often involved restrictions on the form of propositions, but such restrictions unnecessarily limit the allowed axioms while failing to preclude all paradoxes.

Kurt Gödel's first incompleteness theorem in 1931 [3] showed that CL is unsound for any domain more complex than the integers. In particular, certain domains have uncountably many and therefore unknowable (as opposed to unknown) axioms. Such axioms are neither true nor false and thus violate the law of the excluded middle which is a tautology of CL. Without the

law of the excluded middle many traditional proofs are invalid and indirect proof methods are no longer available.

The challenge is to invent a formal deductive logic that is sound for all possible domains and is devoid of paradoxes. That is, its tautologies should be valid, any consistent set of axioms should be allowed regardless of their completeness, and paradoxes should be impossible. More specifically, the logic should be sound for Gödel incomplete domains, should distinguish between concept (sense) and representation (reference), and should ensure that any inconsistency is provable.

3. Solution approach

It is fascinating that many of the problems of CL and other formal logics are not shared by human deductive reasoning (HDR), suggesting that the logic of HDR differs significantly from that of CL. In particular, the man-in-the-street without mathematical training is able to recognize paradoxes (e.g., Does the barber who shaves all men who do not shave themselves shave himself?)

HDR distinguishes humans from other species. It provides a foundation for science, engineering, and mathematics as disciplines unique to humans. It enables sound reasoning within these fields in the absence of complete information (i.e., the logic of HDR is an incomplete logic). There are many other abilities of HDR that are absent from CL, but beyond the scope of this paper. They include abstraction and specialization, generalization, and abilities to exploit and learn from analogy, to ask and answer non yes-no questions, to reason among many domains, and to explain the causes of success and failure in proofs.

The solution approach is to develop a formal deductive logic that emulates aspects of human deductive reasoning (i.e., is intuitive), to show that its tautologies are valid for all possible domains(i.e., logically valid), to ensure that any inconsistency is provable (i.e., precludes paradoxes), and to eliminate common sources of inconsistencies. The latter requires elimination of the traditional law of the excluded middle (which is invalid for Gödel incomplete domains) and a logical distinction between sense (actual examples) and reference (their conceptual representation).

Type theory is not a solution. Russell's theory of types [4] was invented in 1903 to eliminate the famous

$$R = (w|w \notin W)$$

paradox of naive set theory by distinguishing objects, predicates, predicates of predicates, etc. as separate levels of the logic. Its purpose was not to eliminate paradoxes from logic, but to eliminate inconsistencies from set theory. Additionally, stratification of levels provides a non intuitive solution that is at odds with the humanistic approach of PBT logic. And finally, type theory is unsound for domains more complex than integers (i.e., a problem that was not recognized until 1931).

Neither is Zermelo-Fraenkel set theory a solution. It is a first-order logic that was first developed by Ernst Zermelo in 1908 [5]. It too is aimed at eliminating inconsistencies in set theory, does not generally preclude paradoxes, and does not address Gödel incomplete domains.

PBT logic is a form of category theory. Objects in a category theory are unique only up to an isomorphism. What matters is how things are related to one another. The actual nature of

the elements (examples) constituting a category (concept) is irrelevant providing they share analogous relationships.

4. Foundations of PBT

The logic of property-based types (PBT logic) is founded on two longstanding ideas. First, *the only things that can be known about anything are their relationships to other things.* This idea goes back at least to Aristotle's Laws of Thought. It implies that any representation of knowledge must ultimately resolve to representations of relationships.

Second, *rarely is it possible to know everything about anything.* Time, cost, and accessibility make it impractical to know everything about anything in the physical world. Kurt Gödel's first incompleteness theorem showed that it is theoretically impossible to know everything about anything more complex than integers. Thus, PBT logic is an incomplete logic in which it is possible to prove anything that is entailed by what is known, but truths that depend on inaccessible knowledge are unprovable.

Furthermore, for some domains (including all physical domains), it is impossible to represent individual things both completely and accurately. To insure accuracy, PBT logic only characterizes individual things by their known properties, but does not represent or name individuals apart from their properties. Beyond these constraints, the design of PBT logic is influenced primarily by observations about HDR and natural language.

5. Primitive concepts of PBT

A key innovation of PBT logic is its knowledge representation. Things (examples) are isolated from their representations (types). This two level structure makes PBT an intensional logic and allows PBT logic to be a pure calculus of types. It provides the descriptive benefits of self reference without the disadvantage of certain inconsistencies.

PBT logic involves three primitive concepts: properties, types, and examples. Intuitively, a *property* is a characteristic of something. It is analogous to predicates in classical logic and natural languages. It can be represented by a truth-valued function of one free variable. An individual thing is said to satisfy a property if and only if the value resulting from applying the property to the thing is true.

A *type* (property-based type) is a description of a thing in terms (of a subset) of the thing's properties. Each type is analogous to a set of axioms in classical logic. A type can be represented by a finite set of properties where the properties share the same free variable. In PBT logic, types are all and only finite sets of properties. Property-based types correspond to concepts of the mind in human deductive reasoning.

An *example* of a type is anything that satisfies every property that is a member of the type. If the member properties are inconsistent, the type cannot have examples. If a type's member properties are consistent, the type will have at least one example. If its member properties are axiomatically incomplete, it will have multiple examples. That is, each type characterizes a category of things (its examples) that share certain properties (its member properties). Note that an example is an example of every type that comprises a subset of the type's member properties.

PBT logic is a pure calculus of types. It answers questions about categories of individual examples, but individual examples are not represented within its language or proof process. Neither does it distinguish between types and properties. Every type is equivalent to the property that is the logical conjunction of its member properties. Consequently, types, like properties, can be interpreted as truth-valued functions of one free variable. Because examples are not represented, the value of an application of a property to a type is interpreted as true if and only if every example of the type satisfies the property. In the spirit of Gottlob Frege's *Begriff zweiter Stufe* [6], types in PBT logic can be limited by their number of examples. Consistent with the rest of PBT, such constraints yield a subtype rather than a set of examples. For example, five dogs is represented by a type that has five examples as well as the properties of dogs.

Types in PBT logic are defined much as sets of axioms are formed in CL. The primary difference is that while individual things (and sometimes propositions) are named in CL, PBT only names types. Because the number of types is the power set of the number properties, computed types need not be named.

A type of a thing (i.e., an individual object or example) is the conjunction of any subset of the thing's properties. If the type is axiomatically incomplete, it defines the category of all things that satisfy its properties. Otherwise, it defines a type with only one example.

6. Lattice of types

The types of PBT logic form a lattice where the top node is the empty set of properties and the bottom node is the set of all possible properties. Types between the top and bottom nodes are sets of increasing numbers of properties with every type being a subset of all types lower in the lattice. Types between the top and bottom nodes have decreasing numbers of examples (that satisfy the increasing numbers of properties) with the examples of every type being a subset of the examples of all types higher in the lattice. Figure 1 shows an example of the type lattice. Put another way, properties inherit down the lattice and examples inherit up the lattice. Figure 1 shows inheritance of properties and examples among PBT types.

The type called *everything* is a consistent type. It is represented by the empty set of properties and is the top node of the lattice. Every type is a subtype of the everything type. Every thing that is possible or imaginable (i.e., self-consistent) satisfies every property in the everything type (i.e., the empty set). Thus, every thing is an example of everything.

Nothing is an inconsistent type. Its canonical representation is the set of all possible properties and is the bottom node of the lattice. Nothing is a subtype of every type. No thing that is possible or imaginable satisfies every property in the nothing type. Thus, there is nothing that is an example of nothing.

Types between everything and nothing in the lattice may be consistent or inconsistent, but no consistent type is below an inconsistent type. Every consistent type has at least one example. No inconsistent type has an example. As one moves down the lattice, the number of properties increases and the number of examples that satisfy those properties decreases.

Type type is the type of all types. It is a consistent type whose properties include being a conjunction of properties. It is represented by a set of properties that is equivalent to the characteristics of property-based types as described in this paper. By definition, all and only

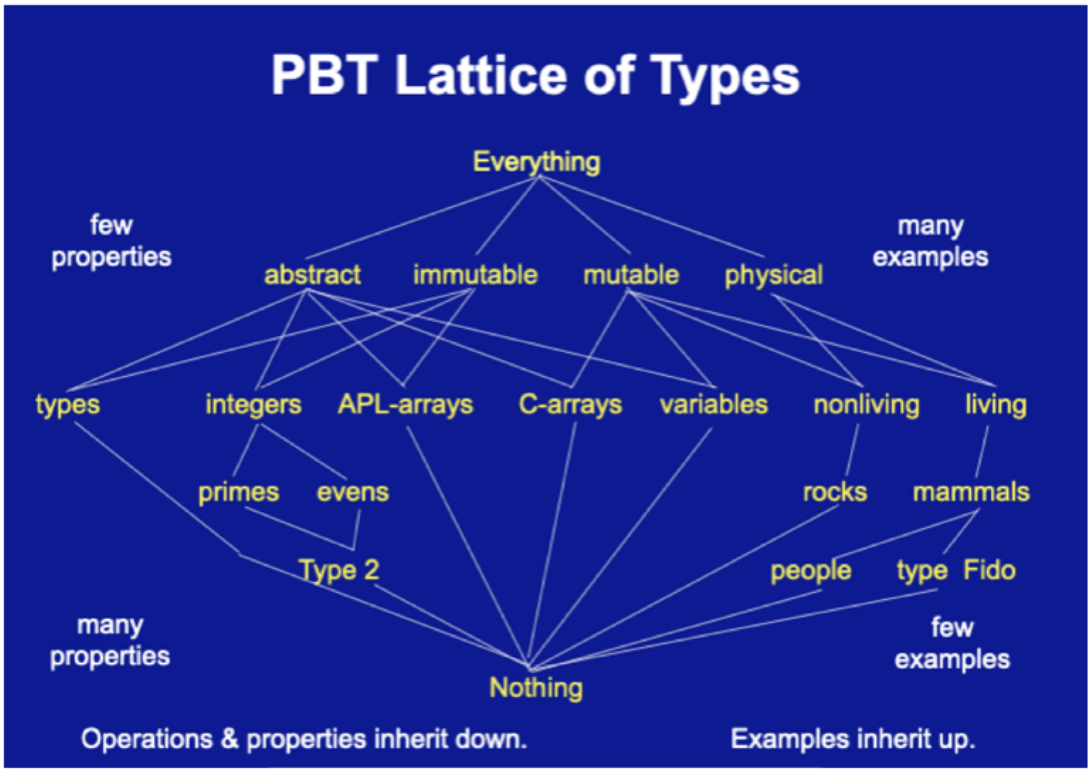

Figure 1: PBT Lattice of Types

types satisfy all properties of type type. The examples of type type are exactly the types of PBT logic. Note also that type type is an example of type type. Because types, but not examples, are artifacts of the logic, that type type is an example of itself is neither a definitional or logical circularity.

Everything and nothing are examples of everything. They serve as the canonical consistent and inconsistent types, respectively. Their roles in PBT logic are analogous to true and false in classical logic. In PBT logic, unlike classical logic, inconsistent types are not equivalent to each other. There is, however, a logical connective (τ) that returns everything or nothing depending on whether its argument is consistent or inconsistent.

7. Structure of the logic system

As a deductive logic, PBT comprises a knowledge representation, a set of tautologies, and a set of axioms. All knowledge in PBT logic is represented by a collection of property based types (types) where each type (as defined in chapter 5) is a conjunction of properties where each property is a predicate. Individual things are not represented, but are instead characterized by examples of the types. A formal symbolic notation (similar to that of Lisp m-expressions and CL notation) can be used to express any PBT type.

The tautologies are rules that specify the functions (logical connectives) and relationships among types and functions on types. The tautologies of PBT logics are similar to those of CL with a few important exceptions. Each rule is interpreted as a statement that is consistent with everything else that is known rather than a true statement. The operator τ indicates that P is true rather than consistent (ie., τP $\leftrightarrow$ (P $\wedge$ $\neg$ τ $\neg$P). The weak law of the excluded middle, P$\vee\neg$P, is syntactically identical to the LEM of CL, but says only that either P or $\neg$P is consistent. The strong law of the excluded middle, τP$\vee\neg\tau$P, is semantically analogous to the LEM of CL, but is less often satisfied. The criticism that without strong LEM certain indirect proofs in CL are precluded, is not justified because only unsound proofs are excluded. The abstraction operation, P$\cap$Q, is added as the intersection of the properties of P and Q. The specialization operation, P$\cup$Q, is added as the union of the properties of P and Q. There are no tautologies involving individuals. The value of the indefinite exemplification operation, N$\in$P, is a type with N arbitrary examples selected from examples of P (e.g., 5$\in$dogs would be interpreted as the type of (any) five dogs). The operator, ϵ, is the definite form of $\in$ (e.g., 5ϵdogs would be interpreted as the type of the five dogs).

The axioms are rules that specify the functions and relationships within and among the domain(s) of application of the logic. They vary from application to application. PBT logic can accommodate multiple domains because all types are subtypes of the universal type, *everything*. Furthermore, because every type is an example of type type, any theorem about a type (as opposed to about its examples) can be proven within the context of type type. Finally, we note that the tautologies of PBT logic are, in fact, the axioms of type type.

8. Human deductive reasoning

Human deductive reasoning (HDR) has served as a primary guide in the design of PBT logic. The logic of HDR is sound for all domains including domains with unknowable properties. It is also intensional, monotonic, devoid of paradoxes, and has a scalable proof process. The existence of HDR serves as proof that such a logic is possible and thus a reasonable expectation for PBT. HDR has at least a 500,000 year advantage on the evolution and refinement of its design, making it unlikely that any current attempt can do better.

Reasoning has both stochastic and deterministic aspects. Current interest in artificial intelligence (AI) has created certain confusions about HDR. The stochastic aspects of reasoning involve pattern recognition and statistical inference. They are shared by all living things. They enable survival of species by marginalizing anomalies. They obtain new knowledge from their environment. Their methods are similar to those of machine learning (AI-ML).

The deterministic aspects of reasoning involve language and deductive inference. They are unique to intelligent species. They enable science, engineering and mathematics by exploiting anomalies. They obtain new knowledge by entailment from what is already known. Their methods are similar to those of formal deductive logic. The deterministic aspects of reasoning constitute human deductive reasoning (HDR).

The importance of emulating HDR is illustrated by the table below which compares AI-ML, CL, type theory(TT), PBT, and HDR. Certain entries for PBT and HDR are arguable.

	AI-ML	CL	TT	PBT	HDR
Stochastic	yes	no	no	no	no
Valid inference rules	no	no	no	yes	yes
Sound for domains with LEM	no	yes	yes	yes	yes
Sound for all domains	no	no	no	yes	yes
Intensional	no	no	no	yes	yes
Monotonic	no	yes	yes	yes	yes
Devoid of paradoxes	—	no	yes	yes	yes
Scalable proof process	no	no	no	yes	yes

9. Formal notation of PBT logic

9.1. Vocabulary

The vocabulary of PBT comprises punctuation symbols, identifiers, and operator symbols.

The punctuation symbols are { } [ ] . ; : :: and λ. Their use is defined in the Grammar section below. Their use is defined in the Grammar section below. An *identifier* (id) is a juxtaposed sequence of letters and digits beginning with a letter.

Depending on the context of the use of an identifier, the identifier names an expression, the expression's value which is always a type, or the set of all examples of the type.

A *number* (natNum) is a juxtaposed sequence of digits. It represents a natural number. A number is axiomatically defined to have the properties of natural numbers and is used to specify the number of examples of a quantified type.

The operator symbols (op) include $\neg \wedge \vee \rightarrow \leftrightarrow \top \tau \cap \cup \sim + - \times \div < \leq = \neq \geq$ and >. An operator names a function that is normally defined by the axioms of the logic. The first eight operators, however, name logical connectives as explained in the Logical Connectives section below.

9.2. Grammar

The formal notation defined here is syntactically unambiguous, but has the expressive power of the full language of PBT logic. The full language grammar (like natural languages) is syntactically ambiguous, is only lightly dependent on punctuation, does not require use of symbols, and instead depends on a semantic grammar using parts-of-speech (i.e., grammatical productions) to categorize words and phrases.

Where practical, the notation here is syntactically similar to that of classical logic. Its use of brackets and failure to syntactically distinguish expressions by their result types gives the notation some of the look and feel of m-expressions in John McCarthy's original Lisp language [7]. The grammar is defined using an extended Backus-Naur form (BNF) [8] that includes { f } to mean zero or more repetitions of a grammatical form f. Semantically, the notation has many analogies in natural languages.

Syntactically, each expression is either simple (sExp), quantified (nExp), or complex (cExp). Semantically, every expression names a type, a predicate, and a category. Also, any function,

proposition, or individual can be represented by an expression. This unification is analogous to the use of any word as a noun in a natural language.

sExp ::= id | natNum | "{" cExp "}" | op nExp.

nExp ::= sExp | natNum nExp | "the" nExp.

sExp ::= "[" cExp { ";" cExp } "]".

cExp ::= nExp | cExp nExp | cExp op nExp.

nExp ::= "λ" sExp cExp | id ":" nExp | id "|" nExp

Special purpose forms are provided for explicitly defining functions, specifying open-form expressions, and local naming of expressions. A lambda expression (λ t P) defines a function with a free variable of type t to be the value of the expression P with the actual parameter substituted for all occurrences of the free variable within P. The *such that* form (X | P) expresses an open-form calculation as the value of the identifier X within the expression P (e.g., X | X×X = 2 defines the square root of 2). The form (X: P) defines X as a local name of expression P as might be used for naming formal parameters or common subexpressions.

prop ::= cExp ".".
soa ::= prop {prop}.
quest :: qExp "?".

Propositions (prop), questions (quest), and sets of axioms (soa) are the sentential forms of the formal PBT grammar. Respectively, they are interpreted as assertions or assumptions, sets of propositions defining a logic, and questions (of one free variable) to be answered. The answer to a question is a simplified version of the expression (cExp), preferably without a free variable. Answering a question invokes the proof process and may involve all of the tautologies and axioms of the logic.

Semantically, the notation is consistent with many features of natural languages. Concepts in natural languages correspond to types in PBT logic. Nouns correspond to named types. Modifiers including adjectives, adverbs, and prepositional phrases correspond to named type-valued functions. Natural language phrases correspond to applications of modifier functions to types. Declarative sentences and clauses correspond to applications of type-valued functions. Conjunction words correspond to logical connection operators such as ∧ and ∨.

10. Paradoxes

PBT logic precludes paradoxes by naming and referencing only types, but interpreting all references as references to examples. This allows a syntactic self reference without a semantic cycle. In this way, PBT logic is similar to a second order logic without paradoxes. While the

set of all naive sets that do not contain themselves is paradoxical in CL (and inconsistent), the category of all types that do not exemplify themselves is well defined (and consistent).

11. Incompleteness

To be applicable to any possible domain a logic must be both incomplete (i.e., one that does not require a complete axiomatic description) and sound. A domain is possible if and only if its axioms are consistent. The primary barrier to a sound logic for Gödel incomplete domains is the loss of the law of the excluded middle (LEM) (i.e., every proposition is either true or false), which has been obvious since the time of Aristotle, but is nevertheless false. PBT has a slightly weaker LEM that says every type (i.e., finite set of properties) is either consistent or inconsistent. Because Gödel's second incompleteness theorem is about infinite domains rather than finite sets of axioms, it does not apply to PBT logic. A type typically characterizes a domain by a subset of the domain's properties. In a sound logic, if the type is inconsistent, it describes nothing. If it is consistent, there will be real or imaginable domains that satisfy all properties of the type and any conclusions involving that type will be valid for all such domains. Incompleteness can make indirect proofs more difficult. To prove that $P \wedge Q$ is true both $P \wedge Q$ and $\neg(P \wedge \neg Q)$ must be consistent. That $P \leftrightarrow \neg\neg P$ in PBT logic simplifies all proofs.

12. Conclusions

The logic of property-based types (PBT logic) as outlined here provides solutions to several previously unsolved problems of classical logic. It is a pure calculus of types (analogous to concepts of the human mind) and modeled after human deductive reasoning. It has advantages of type theory, but is intuitive with a radically different structure. It is a deductive logic that is sound for any domain and in which any inconsistency can be proven.

It is hoped that PBT logic will (a) stimulate research toward logics that more closely replicate the logic of human deductive reasoning, and (b) provide a formal foundation for a new class of logic programming languages.

Acknowledgments

The development of PBT logic to this stage has been a long and often difficult effort. Throughout the effort Professor Thomas A Standish of the University of California - Irvine has made many helpful suggestions and always encouraging. His assistance is greatly appreciated.

References

[1] C. Shields, Aristotle, The Stanford Encyclopedia of Philosophy (Spring 2022 Edition), Edward N. Zalta, (ed.) (2022).

[2] J. van Heijenoort, From Frege to Gödel: A Source Book in Mathematical Logic, 1879-1931, Cambridge and London: Harvard University Press., 1967.

[3] P. Raatikainen, Gödel's incompleteness theorems, The Stanford Encyclopedia of Philosophy (Spring 2022 Edition), Edward N. Zalta, (ed.) (2022).
[4] B. Russell, The Principles of Mathematics, Cambridge: Cambridge University Press., 1903.
[5] E. Zermelo, Untersuchungen über die grundlagen der mengenlehre, i, Mathematische Annalen 65 (1908) 261–281.
[6] D. Westerståhl, Generalized quantifiers, The Stanford Encyclopedia of Philosophy (Winter 2019 Edition), Edward N. Zalta, (ed.) (2019).
[7] J. McCarthy, Recursive functions of symbolic expressions and their computation by machine, part i, Communications of the ACM 3 (1960) 184–195.
[8] J. W. Backus, The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM conference, Proceedings of the International Conference on Information Processing, UNESCO (1959) 125–132.

A. Logical connectives

In PBT logic, a logical connective is a type-valued function on types. The connectives $\wedge$, $\vee$ and $\neg$ are defined by their effect on examples and are analogous to those of CL. The connectives $\cap$, $\cup$ and $\sim$ are defined by their effect on properties and do not have analogies in CL. The connectives include:

Conjunction ($\wedge$)	P$\wedge$Q is the intersection of examples of P and Q
Disjunction ($\vee$)	P$\vee$Q is the union of examples of P and Q
Abstraction ($\cap$)	P$\cap$Q is the intersection of properties of P and Q
Specialization ($\cup$)	P$\cup$Q is the union of properties of P and Q
Negation ($\neg$)	$\neg$P is a type that is inconsistent when P is consistent and vice-versa
Implication ($\rightarrow$)	P$\rightarrow$Q is everything if P is a subtype of Q and otherwise nothing
Consistent (τ)	τP is everything if P is consistent and otherwise nothing