For Cybersecurity, Computer Science Must Rely on the Opposite of Gödel's Results

Carl Hewitt†

Abstract—**This article shows how fundamental higher-order theories of mathematical structures of computer science (e.g. natural numbers [Dedekind 1888] and Actors [Hewitt et. al. 1973]) are cetegorical meaning that they can be axiomatized up to a unique isomorphism thereby removing any ambiguity in the mathematical structures being axiomatized. *Having these mathematical structures precisely defined can make systems more secure because there are fewer ambiguities and holes for cyberattackers to exploit.* For example, there are no infinite elements in models for natural numbers to be exploited. On the other hand, the 1st-order theories of Gödel's results necessarily leave the mathematical structures ill-defined, e.g., there are necessarily models with infinite integers.**

Cyberattackers have severely damaged national, corporate, and individual security as well causing hundreds of billions of dollars of economic damage. *A significant cause of the damage is that current engineering practices are not sufficiently grounded in theoretical principles*. In the last two decades, little new theoretical work has been done that practically impacts large engineering projects with the result that computer systems engineering education is insufficient in providing theoretical grounding. If the current cybersecurity situation is not quickly remedied, it will soon become much worse because of the projected development of Scalable Intelligent Systems by 2025 [Hewitt 2019].

Gödel strongly advocated that the Turing Machine is the preeminent universal model of computation. A Turing machine formalizes an algorithm in which computation proceeds without external interaction. However, computing is now highly interactive, which this article proves is beyond the capability of a Turing Machine. *Instead of the Turing Machine model, this article presents an axiomatization of a universal model of digital computation (including implementation of Scalable Intelligent Systems) up to a unique isomorphism.*

Index Terms—**categorical theories, strong types, Scalable Intelligent Systems, Alonzo Church, Kurt Gödel, Richard Dedekind**

I. Introduction

The approach in this article is to embrace *all* of the most powerful tools of classical mathematics in order to provide mathematical foundations for Computer Science. Fortunately, the results presented in this article are technically simple so they can be readily automated, which will enable better collaboration between humans and computer systems.

Mathematics in this article means the precise formulation of standard mathematical theories that axiomatize the following standard mathematical structures up to a unique isomorphism: Booleans, natural numbers, reals, ordinals, sets, computable procedures, and Actors, as well as the theories of these structures.

In a strongly typed mathematical theory, every proposition, mathematical term, and program expression has a type where there is no universal type ~~*Any*~~. Types are constructed bottom up from mathematical types that are individually categorically axiomatized in addition to the types of a theory being categorically axiomatized as a whole.

[Russell 1906] introduced types into mathematical theories to block paradoxes such as *The Liar* which could be constructed as a paradoxical fixed point using the mapping p↦¬p, *except for the requirement that each proposition must have an order beginning with 1st-order*. Consequently, the mapping p↦¬p has no fixed point because ¬p has order one greater than the order of p because p is a *propositional variable*, as in the lambda calculus version of the mapping, i.e., λ[p]¬p. **Thus because of orders on propositions, there is no paradoxical fixed point for the mapping** p↦¬p **which *if it existed* could be called** *I'mFalse* **such that** ***I'mFalse*⇔¬*I'mFalse*.** Unfortunately in addition to attaching orders to propositions, [Whitehead and Russell 1910-1913] also attached orders to the other mathematical objects (such as natural numbers), which made the system unsuitable for standard mathematical practice.

II. Limitations of 1st-Order Logic

Wittgenstein correctly proved that allowing the proposition *I'mUnprovable* [Gödel 1931] into mathematics [Russell and Whitehead 1910-1913] infers a contradiction as follows:

"Let us suppose [Gödel 1931 was correct and therefore] I prove the improvability (in Russell's system) of [Gödel's *I'mUnprovable*] *P*; [*i.e.*, $\vdash_{\mathbb{Russell}} \nvdash_{\mathbb{Russell}} P$ where $P \Leftrightarrow \nvdash_{\mathbb{Russell}} P$] then by this proof I have proved *P* [*i.e.*, $\vdash_{\mathbb{Russell}} P$ because $P \Leftrightarrow \nvdash_{\mathbb{Russell}} P$]. Now if this proof were one in Russell's system [*i.e.*, $\vdash_{\mathbb{Russell}} \vdash_{\mathbb{Russell}} P$] — I should in this case have proved at once that it belonged [*i.e.*, $\vdash_{\mathbb{Russell}} P$] and did not belong [*i.e.*, $\vdash_{\mathbb{Russell}} \neg P$ because $\neg P \Leftrightarrow \vdash_{\mathbb{Russell}} P$] to Russell's system. But there is a contradiction here! [*i.e.*, $\vdash_{\mathbb{Russell}} P$ and $\vdash_{\mathbb{Russell}} \neg P$] … **[This] is what comes of making up such propositions**." [emphasis added] [Wittgenstein 1978]

†C. Hewitt is the Board Chair of iRobust (International Society for Inconsistency Robustness) and an emeritus professor of MIT. His homepage is https://professorhewitt.blogspot.com/

Gödel made important contributions to the metamathematics of 1st-order logic with the countable compactness theorem and formalization of provability [Gödel 1930]. However decades later, Gödel asserted that the [Gödel 1931] inferential undecidability results were for a 1st-order theory (e.g. like [Paulson 2014]) instead of the theory ℝ𝕦𝕤𝕤𝕖𝕝𝕝 [Russell and Whitehead 1910-1913] as originally stated in [Gödel 1931]. In this way, **Gödel dodged the point of Wittgenstein's criticism and instead asked if Wittgenstein was "crazy."**

Technically, the result in [Gödel 1931] was as follows:

$$\text{Consistent}[\mathbb{Russell}] \Rightarrow \vdash_{\mathbb{Russell}} \nvdash_{\mathbb{Russell}} P$$

where $P \Leftrightarrow \nvdash_{\mathbb{Russell}} P$ and Consistent[ℝ𝕦𝕤𝕤𝕖𝕝𝕝] if an only if there is no proposition Ψ such that $\vdash_{\mathbb{Russell}} \Psi \wedge \neg \Psi$, However, Wittgenstein was understandably taking it as a given that ℝ𝕦𝕤𝕤𝕖𝕝𝕝 is consistent because it formalized standard mathematical practice and had been designed to block known paradoxes (such as *The Liar*) using orders on propositions. Consequently, Wittgenstein elided the result in [Gödel 1931 to $\vdash_{\mathbb{Russell}} \nvdash_{\mathbb{Russell}} P$. His point was that ℝ𝕦𝕤𝕤𝕖𝕝𝕝 is consistent provided that the proposition $\vdash_{\mathbb{Russell}} \nvdash_{\mathbb{Russell}} P$ is **not** added to ℝ𝕦𝕤𝕤𝕖𝕝𝕝. Wittgenstein was justified because the standard theory of natural numbers is arguably consistent because it has a model. [Dedekind 1888]

Although 1st-order propositions can be useful (e.g. in 1st-order proposition satisfiability testers), 1st-order theories are unsuitable as the mathematical foundation of computer science for the following reasons:

- **Compactness** Every 1st-order theory is compact [Maltsev 1936] (meaning that every inconsistent set of propositions has a finite inconsistent subset), which is false of the standard theory of natural numbers for the following reason: if k is a natural number then the set of propositions {i>k | i is a natural number} is inconsistent but has no finite inconsistent subset, thereby contradicting compactness.
- **Monsters** Every 1st-order theory is ambiguous about fundamental mathematical structures such as the natural numbers, lambda expressions, and Actors [Hewitt and Woods assisted by Spurr 2019]. For example,
  - Every 1st-order axiomatization of the natural numbers has a model with an element (which can be called ∞) for a natural number, which is a "monster" [Lakatos 1976] because ∞ is larger than every standard natural number.
  - Every 1st-order theory $\mathbb{T}$ that can formalize its own provability has a model M with a Gödelian "monster" element Γ such that $\models_M \vdash_{\mathbb{T}} \Gamma \wedge \neg \Gamma$ because according to [Gödel 1931] $\nvdash_{\mathbb{T}}$ Consistent[$\mathbb{T}$] and consequently because of the 1st-order model "completeness" theorem [Gödel 1930] there must be some model of $\mathbb{T}$ in which Consistent[$\mathbb{T}$] is false. [cf. Artemov 2019]

  Such monsters are highly undesirable in models of standard mathematical structures in Computer Science because they are inimical to model checking.
- **Inconsistency** This article shows that a 1st-order theory that can formalize its own provability is inconsistent.
- **Intelligent Systems.** If a 1st-order theory is not consistent, then it is useless because each and every proposition (no matter how nonsensical) can be proved in the theory. However, Scalable Intelligent Systems must reason about massive amounts of pervasively-inconsistent information. [Hewitt and Woods assisted by Spurr 2019] Consequently, such systems cannot always use 1st-order theories. Conversational Logic [Hewitt 2016-2019] needs to be used to reason about inconsistent information in Scalable Intelligent Systems. [cf. Woods 2013]

Consequently, Computer Science must move beyond 1st-order logic for its foundations.

III. Strong Types

Types must be strong to prevent inconsistency but flexible to all valid inference. (See appendix on how known paradoxes are blocked.) *Although mathematics in this article necessarily goes beyond 1st-order logic, standard mathematical practice is used. Wherever possible, previously used notation is employed.* The following notation is used for types:

- The notation x:t means that x is of type t. For example, 0:N expresses that 0 is of type N, which is the type of a natural number. Types are *intension*, i.e., if x:t_1 and x:t_2 for every x does not mean that t_1=t_2 where t_1 and t_2. are types.
- $t_2^{t_1}$ is type of *all* functions from t_1 into t_2 where t_1 and t_2. are types. A function is total and may be *uncomputable*. For example, N^N is the type all total functions from natural numbers into the natural numbers, which are *uncountable*. If f:N^N, then f[3] is the value of function f on argument 3.
- $t_1 \rightarrow t_2$ is type of *computable* procedures from t_1 into t_1 where t_1 and t_2 are types. A computable procedure can be partial and can be indeterminate in its outcome. For example, $[N] \rightarrow N$ is the type all partial procedures of one argument (which is a natural number) into the type of natural numbers. If p:$[N] \rightarrow N$, then p▪[3] starts a computation by providing input [3] to procedure p. *It might happen that* p▪[3] *does not return a value*. Similarly anAccount▪deposit[$5] sends a deposit message to the Actor anAccount.
- $[t_1, t_2]$ is type of pairs of t_1 and t_2 where t_1 and t_2 are types. For example, $[N,$ *Boolean*$]$ is the type of pairs whose first is a natural number and whose second is a Boolean.
- *TypeOf*◁t▷ is type of t where is a type. For example, N:*TypeOf*◁N▷ meaning that N is of type *TypeOf*◁N▷ producing an infinite hierarchy of types of types somewhat like the hierarchy of universes in [Martin-Löf 1998]. There is no type ~~*Type*~~ thereby blocking Girard's paradox [Girard 1972, Martin-Löf 1998].
- *PropositionOfOrder*◁i▷ is type of a proposition of order i where i:N_+ and N_+ is the type of positive natural

numbers. For example, *PropositionOfOrder*◁1▷ is the type of propositions of order 1.

- t∋P is type of t restricted to P where t is a type and P:*PropositionOfOrder*◁i▷^{t}. For example replacement for types (range of a function f:$t_2^{t_1}$ is t_2∋λ[y:t_2] ∃[x:t_1] y=f[x]

Types are constructed bottom-up from types that categorically axiomatized up to a unique isomorphism. *Type checking is linear in the size of the propositon, mathematical term or procedural expression to be type checked.* **See appendix for syntax of propositions, mathematical terms, and procedural expressions.**

IV. Standard Theories of Computer Science

Cybersecurity requires that fundamental mathematical structures in Computer Science must be precisely defined. This section shows how to precisely define natural numbers**. It is followed by a section on how to precisely define Actors, which are the fundamental abstraction of computation.**

The mathematical theory ℕat that axiomatises the Natural Numbers has the following axioms building on [Dedekind 1888]:

- 0:N // 0 is of type N
- $+_1$:N^N // $+_1$ (successor) is of type N^N
- ∄[i:N] $+_1$[i]=0 // 0 is not a successor
- ∀[i,j:N] $+_1$[i]=$+_1$[j] ⇨ i=j // $+_1$ is 1 to 1

In addition, ℕat has the following induction axiom, **which has uncountable instances**:

∀[i:N_+, P:*PropositionOfOrder*$_{\text{NAT}}$◁i▷^{N}]
(P⟦0⟧ ∧ ∀[i:N] P⟦i⟧⇨P⟦$+_1$[i]⟧) ⇨ ∀[i:N] P⟦i⟧

For example, if P is a 1st-order predicate, i.e., P:*PropositionOfOrder*$_{\text{NAT}}$◁1▷^{N} then, the following is a proposition of order 2:

(P⟦0⟧ ∧ ∀[i:N] P⟦i⟧⇨P⟦$+_1$[i]⟧) ⇨ ∀[i:N] P⟦i⟧

Note that the above proposition can be used in an induction axiom of order 2 to produce a proposition of order 3, etc.

𝕄eta◁ℕat▷

𝕄eta◁ℕat▷ is a meta theory of ℕat for proving theorems about ℕat which ***directly* expresses provability of a proposition** Ψ **in** ℕat **using** $\vdash_{\mathbb{N}\text{at}}$Ψ instead of using Gödel numbers because there are not enough Gödel numbers to represent all uncountably many propositions that are instances of the induction axiom.

Procedures of ℕat

Eval◁t▷:[*Expression*◁t▷ **in** *Environment*]→t is a procedure [McCarthy et. al. 1962] that corresponds to a universal Turing machine [Church 1936] as follows:

- Eval◁t▷■[x:*Expression*◁t▷] ≡
  Eval◁t▷■[x:*Expression*◁t▷ **in** EmptyEnvironment]
- Eval◁t▷l■[x:*Identifier*◁t▷ **in** e:*Environment*] ≡
  Lookup[x **in** e]
- Eval◁$t2$▷■[operator■operand **in** e:*Environmment*] ≡
  (Eval◁$t1$→$t2$▷■[operator **in** e])■(Eval◁$t1$▷■[operand **in** e])
  // apply the value of operator to
  // the value of operand
- Eval◁$t1$→$t2$▷■[(λ x1 body) **in** e:*Environmment*] ≡
  λx2:$t1$ Eval■[body **in** Bind■[x1 **to** x2 **in** e]]
  // eval body in a new environment with x1 bound
  // to x2 as an extension of e

In addition, the lambda calculus has the primitive *Fix* such that
∀[F:*Functional*◁t_1,t_2▷] ***Fix***◁t_1,t_2▷■[F]**=** F■[*Fix*◁t_1,t_2▷■[F]]
where *Functional*◁t_1,t_2▷ ≡ *[[*t_1*]→*t_2*]→([*t_1*]→*t_2*)*

Proof Checkers in ℕat

A proof checker pc:*ProofChecker*◁ℕat▷ is a provably total boolean-valued procedure of two arguments that checks if the second argument is validly inferred from the first argument. The following notation (which is part of the theory ℕat) means that pc is proof checker such that proposition Ψ_1 infers proposition Ψ_2 in ℕat: $\Psi_1 \vdash^{\text{pc}}_{\text{Nat}} \Psi_2$ such that:

∀[Ψ_1,Ψ_2:*Proposition*◁ℕat▷]
($\Psi_1 \vdash_{\mathbb{N}\text{at}} \Psi_2$) ⇔ ∃[pc:*ProofChecker*◁ℕat▷] $\Psi_1 \vdash^{\text{pc}}_{\text{Nat}} \Psi_2$

Proof checking in ℕat is computationally decidable because:

∀[Ψ_1,Ψ_2:*Proposition*◁ℕat▷], pc:*ProofChecker*◁ℕat▷]
($\Psi_1 \vdash^{\text{pc}}_{\text{Nat}} \Psi_2$) ⇔ pc■[$\Psi_1$,$\Psi_2$]=True

where pc■[Ψ_1,Ψ_2] means the invocation of procedure pc with arguments Ψ_1 and Ψ_2. For example, there is a Chaining for Inference checker such that if Ψ_1 is Ψ⋀ (Ψ$\vdash^{\text{pc}}_{\text{Nat}}$ Φ), then ChainingForInferenceChecker■[Ψ_1,Ψ_2]=True if Ψ_2=Φ and pc■[Ψ,Φ]=True, otherwise pc■[Ψ_1,Ψ_2]=False as follows**:**

ChainingForInferenceChecker■[Ψ_1,Ψ_2] ≡
Ψ_1 if Ψ$\vdash^{\text{pc}}_{\text{Nat}}$ Φ then Ψ_2=Φ and pc■[Ψ,Φ]=True**,**
else False

The proof checker for the induction axiom is as follows:

InductionChecker■[Ψ,Ψ_2] ≡
Ψ_1 if (P⟦0⟧ ∧ ∀[i:N] P⟦i⟧⇨P⟦$+_1$[i]⟧)
then Ψ_2 = ∀[i:N] P⟦i⟧**,**
else False

Note that InductionChecker correctly checks uncountably many instances of each of the ℕat induction axioms.

There are uncountable proof checkers in ℕat **which is made possible because proof checkers can operate on higher order types, e.g., they are not restricted to strings.** For example, there are *uncountable* proof checkers of the form ForAllEliminationChecker◁t▷[c] where t is a type and c:t such that

ForAllEliminationChecker◁t▷[c]■[Ψ_1,Ψ_2] ≡
Ψ_1 if (∀[x:t] P[x]) then Ψ_2=P[c], else False

Consequently,

$$(\forall[x{:}t]\ P[x]) \vdash^{\text{ForAllEliminationChecker}\triangleleft t\triangleright[c]}_{\text{Nat}} P[c]$$

Inferential soundness means that a theorem in ℕat can be used in proofs in ℕat. A consequence of Infrential Soundness is that unrestricted cut-elimination does not hold for ℕat.

Theorem: Inferential Soundness of ℕat, i.e.,

$$\vdash_{\text{Meta}\triangleleft\text{Nat}\triangleright} \forall[\Psi{:}\mathit{Proposition}\triangleleft\mathbb{N}\text{at}\triangleright]\ (\vdash_{\text{Nat}} \Psi) \vdash_{\text{Nat}} \Psi$$

*Proof***.** Follows immediately from the rule TheoremUse, i.e., $(\vdash_{\text{Nat}} \Psi) \vdash^{\text{TheoremUse}}_{\text{Nat}} \Psi$

Theorem: Deduction for ℕat, i.e.,

$$\vdash_{\text{Meta}\triangleleft\text{Nat}\triangleright} \forall[\Phi,\Psi{:}\mathit{Proposition}\triangleleft\mathbb{N}\text{at}\triangleright]\ (\vdash_{\text{Nat}} \Phi\Rightarrow\Psi) \Leftrightarrow (\Phi\vdash_{\text{Nat}}\Psi)$$

Proof Suppose $\vdash_{\text{Nat}}\Phi\Rightarrow\Psi$ and consequently $\Phi\Rightarrow\Psi$ by Inferential Soundness. Further suppose Φ. Then Ψ by ChainingForImplication and consequently $\Phi\vdash_{\text{Nat}}\Psi$ by InferenceIntroduction.

On the other hand suppose $\Phi\vdash_{\text{Nat}}\Psi$. Further suppose Φ. Then Ψ by ChainingForInference and consequently $\vdash_{\text{Nat}}\Phi\Rightarrow\Psi$ by ImplicationIntroduction.

Theorem Inferential Adequacy, i.e.,

$$\forall[\Psi{:}\mathit{Proposition}\triangleleft\mathbb{N}\text{at}\triangleright]\ (\vdash_{\text{Nat}}\Psi)\Rightarrow\vdash_{\text{Nat}}\vdash_{\text{Nat}}\Psi$$

Proof: Suppose $\vdash_{\text{Nat}}\Psi$. Let $\vdash^{\text{pc1}}_{\text{Nat}} \Psi$ so that pc1▪[Ψ]=<u>True</u>**.** Then a provably total procedure pc2: *ProofChecker*◁ℕat▷ can be defined such that pc2▪[$\vdash^{\text{pc1}}_{\text{Nat}}\Psi$]=<u>True</u> meaning that $\vdash^{\text{pc2}}_{\text{Nat}}\vdash^{\text{pc1}}_{\text{Nat}}\Psi$. Consequently, $\vdash_{\text{Nat}}\vdash_{\text{Nat}}\Psi$.

Theorem Unique Categoricity of ℕat for *N*. [Dedekind 1888]: If *M* be a type satisfying the axioms of ℕat, then there is a unique isomorphism I with *N* defined as follows:

- $I{:}\mathcal{M}^{N}$
- $I[0] \equiv 0_{\mathcal{M}}$
- $I[+_1[j]] \equiv +^{\mathcal{M}}_1[I[j]]$

I is a unique isomorphism because of the following;

- I is defined on *N*
- I is 1-1
- I is onto *M*
- I is a homomorphism
  - $I[0] \equiv 0_{\mathcal{M}}$
  - $\forall[i{:}N]\ I[+_1[j]] \equiv +^{\mathcal{M}}_1[I[j]]$
- I^{-1} is a homomorphism
  - $I^{-1}[0_{\mathcal{M}}] = 0$
  - $\forall[y{:}\mathcal{M}]\ I^{-1}[+^{\mathcal{M}}_1[y]] = +_1[I^{-1}[y]]$
- If g is an isomorphism of *N* with *M*, then g=I

Corollary There are no infinite numbers in models of the theory ℕat, i.e., if *M* satisfies the axiom of ℕat, then

$$\nexists[j{:}\mathcal{M}]\ \forall[i{:}N]\ i < j$$

Computational Undecidability of Provability in ℕat

The predicate Halt can be defined as follows:

$$\text{Halt}\triangleleft N\triangleright[x{:}\mathit{Expression}\triangleleft N\triangleright] \equiv \exists[y{:}N]\ y=\text{Eval}_{\blacksquare}[x]$$

*Theorem***.** Halt is computationally undecidable [Church 1935, Turing 1936], i.e.,

$$\nexists[f{:}[\mathit{Expression}\triangleleft N\triangleright]\rightarrow\mathit{Boolean}]\ \forall[x{:}\mathit{Expression}\triangleleft N\triangleright]\ f_{\blacksquare}[x]=\underline{\text{True}} \Leftrightarrow \text{Halt}\triangleleft N\triangleright[x]$$

Theorem. Whether a proposition is a theorem of ℕat is computationally undecidable [Church 1935, Turing 1936], i.e., $\nexists[f{:}[\mathit{Proposition}\triangleleft\mathbb{N}\text{at}\triangleright]\rightarrow\mathit{Boolean}]$

$$\forall[\Psi{:}\mathit{Proposition}\triangleleft\mathbb{N}\text{at}\triangleright]\ f_{\blacksquare}[\Psi]=\underline{\text{True}} \Leftrightarrow \vdash_{\text{Nat}}\Psi$$

Proof. Follows immediately from the following:

$$\forall[x{:}\mathit{Expression}\triangleleft N\triangleright]\ \text{Halt}\triangleleft N\triangleright[x] \Leftrightarrow \vdash_{\text{Nat}} \text{Halt}\triangleleft N\triangleright[x]$$

Theorem. Indiscernibles, i.e.,

$$\forall[t{:}\mathit{TypeIn}\triangleleft\mathbb{N}\text{at}\triangleright;\ x_1,x_2{:}t]\ (\forall[P{:}\mathit{PropositionOfOrder}\triangleleft 1\triangleright^{t}]\ P[\![x_1]\!]\Leftrightarrow P[\![x_2]\!]])\Rightarrow x_1=x_2$$

*Proof***.** Induction on *TypeIn*◁ℕat▷ using the base case

$$\forall[x_1,x_2{:}N]\ \mathit{Let}\ P=(\lambda[x{:}N]\ x=x_1)\ \mathit{in}\ P[\![x_1]\!]\Leftrightarrow P[\![x_2]\!]])\Rightarrow x_1=x_2$$

Inferential Undecidability of ℕat

Theorem. ℕat is inferentially undecidable, i.e.,

$$\vdash_{\text{Meta}\triangleleft\text{Nat}\triangleright}\exists[\Psi{:}\mathit{Proposition}\triangleleft\mathbb{N}\text{at}\triangleright]\ (\nvdash_{\text{Nat}}\Psi) \wedge (\nvdash_{\text{Nat}}\neg\Psi)$$

Proof Suppose to obtain a contradiction that ℕat is inferentially decidable and consequently

$$\forall[x{:}\mathit{Expression}\triangleleft N\triangleright]\ (\vdash_{\text{Nat}}\text{Halt}\triangleleft N\triangleright[x]) \vee (\vdash_{\text{Nat}}\neg\text{Halt}\triangleleft N\triangleright[x])$$

Since only countably many instances of the natural number induction axiom could have been used in the above proofs, the halting problem is computationally decidable by computationally enumerating the proofs, which is a contradiction.

In practice, computational undecidability of provability and inferential undecidability, do not impose limitations on the ability to prove theorems for mathematical theories of Intelligent Systems.

Definition: P <u>predicateOn</u>$_{\text{NAT}}$ $t\Leftrightarrow$

$$\exists[i{:}N_+]\ P{:}\mathit{PropositionOfOrder}_{\text{Nat}}\triangleleft i\triangleright^{t}$$

The following axioms hold for *TypeIn*◁ℕat▷ (the type of types in ℕat) because types are *intensional*:

- $N{:}\mathit{TypeIn}\triangleleft\mathbb{N}\text{at}\triangleright$
- $\forall[i{:}N_+]\ \mathit{PropositionOfOrder}_{\text{Nat}}\triangleleft i\triangleright{:}\mathit{TypeIn}\triangleleft\mathbb{N}\text{at}\triangleright$
- $\forall[t_1,t_2,t_3,t_4{:}\mathit{TypeIn}\triangleleft\mathbb{N}\text{at}\triangleright]\ [t_1,t_2]=[t_3,t_4] \Rightarrow t_1=t_2 \wedge t_3=t_4$
- $\forall[t1,t2,t3,t4{:}\mathit{TypeIn}\triangleleft\mathbb{N}\text{at}\triangleright]\ t1^{t2}=t3^{t4}\Rightarrow t1=t2 \wedge t3=t4$
- $\forall[t_1,t_2,t_3,t_4{:}\mathit{TypeIn}\triangleleft\mathbb{N}\text{at}\triangleright]\ t_1\rightarrow t_2=t_3\rightarrow t_4 \Rightarrow t_1=t_2 \wedge t_3=t_4$
- $\forall[t_1,t_2{:}\mathit{TypeIn}\triangleleft\mathbb{N}\text{at}\triangleright;$ P1 <u>predicateOn</u>$_{\text{Nat}}$ t_1, P_2 <u>predicateOn</u>$_{\text{Nat}}$ $t_2]$ $t_1\underline{\ni}P_1=t_2\underline{\ni}P_2 \Rightarrow t_1=t_2 \wedge P1=P_2$
- $\forall[t_1,t_2{:}\mathit{TypeIn}\triangleleft\mathbb{N}\text{at}\triangleright]\ \mathit{TypeOf}\triangleleft t_1\triangleright=\mathit{TypeOf}\triangleleft t_2\triangleright \Rightarrow t_1=t_2$

For example, N^{N}:*TypeIn*◁ℕat▷, etc.

The following induction axiom holds, *which has uncountable instances*:

∀[P predicateOn$_{ℕat}$ *TypeIn*◁ℕat▷]
(P⟦N⟧
∧ ∀[t_1,t_2:*TypeIn*◁ℕat▷] P⟦t_1⟧∧P⟦t_1⟧⇨P⟦[t_1,t_2]⟧
∧ ∀[t_1,t_2:*TypeIn*◁ℕat▷] P⟦t_1⟧∧P⟦t_2⟧⇨P⟦$t1^{t2}$⟧
∧ ∀[t_1,t_2:*TypeIn*◁ℕat▷] P⟦t_1⟧∧P⟦t_2⟧⇨P⟦t_1→t_2⟧
∧ ∀[t:*TypeIn*◁ℕat▷, Q predicateOn$_{ℕat}$ t]
P⟦t⟧⇨P⟦t∋Q⟧
∧ ∀[i:N_+] P[*PropositionOfOrder*$_{ℕat}$◁i▷⟧
∧ ∀[t:*TypeIn*◁ℕat▷] P⟦t⟧⇨P⟦*TypeOf*◁t▷⟧)
⇨ ∀[t:*TypeIn*◁ℕat▷] P⟦t⟧

Theorem Unique categoricity of *TypeIn*◁ℕat▷, i.e., if M is a type satisfying the theory ℕat, then there is a unique isomorphism I between *TypeIn*◁ℕat▷ and *TypeIn*$_M$◁ℕat▷ defined as follows:

- I[N] ≡ N_M
- I[[t_1,t_2]] ≡ [I[t_1], I[t_2]]$_M$
- I[$t1^{t2}$] ≡ I[$t1$]$^{I[t2]}$
- I[t_1→t_2] ≡ I[t_1]→ I[t_2]
- I[*TypeOf*◁t▷] ≡ *TypeOf*$_M$◁I[t]▷ I[t∋P] defined by Induction on *TypeIn*◁ℕat▷ using the following cases on t:
  - N, then I[t∋P] ≡ N_M∋$_M$ λ[y] P⟦I^{-1}[y]⟧
  - [t_1,t_2], then I[t∋P] ≡ [I[t_1], I[t_2]]$_M$∋$_M$ λ[y] P⟦I^{-1}[y]⟧
  - $t1^{t2}$, then I[t∋P] ≡ I[$t1$]$^{I[t2]}$]∋$_M$ λ[y] P⟦I^{-1}[y]⟧
  - t_1→t_2, then I[t∋P] ≡ I[I[t_1]→ I[t_2]]∋$_M$ λ[y] P⟦I^{-1}[y]⟧
  - *TypeOf*◁t_1▷, then
    I[t∋P] ≡ *TypeOf*$_M$◁I^{-1}[t_1]▷∋$_M$ λ[y] P⟦I^{-1}[y]⟧
  - t_1∋P$_1$, then I[t∋P] ≡ I[t_1]∋$_M$ λ[y] P⟦I^{-1}[y]⟧∧P$_1$⟦I^{-1}[y]⟧

The following induction axiom holds for propositions of ℕat, *which has uncountable instances*:

((∀[P predicateOn$_{ℕat}$ *Proposition*◁ℕat▷)
∧ (∀[t:*TypeIn*◁ℕat▷; x$_1$,x$_2$:t] P⟦x1=x2⟧
∧ ∀[t_1,t_2:*TypeIn*◁ℕat▷; x:t_2] P⟦x:t⟧
∧ ∀[Ψ:*Proposition*◁ℕat▷] P⟦Ψ⟧⇨P⟦¬Ψ⟧
∧ ∀[Ψ$_1$,Ψ$_2$:*Proposition*◁ℕat▷]
P⟦Ψ$_1$⟧∧P⟦Ψ$_2$⟧⇨P⟦Ψ$_1$∧Ψ$_2$⟧
∧ ∀[t:*TypeIn*◁ℕat▷; Q predicateOn$_{ℕat}$ t]
(∀[x:t] P⟦Q⟦x⟧⟧)⇨P[∀[x:t] Q⟦x⟧]))
⇨ ∀[Ψ:*Proposition*◁ℕat▷] P[Ψ]

Theorem. *Proposition*◁ℕat▷ is characterized up to a unique isomorphism.

Theorem. Instance Adequacy for ℕat, i.e.,

∀[t:*TypeIn*◁ℕat▷, P predicateOn$_{ℕat}$ t]
(∀[x:t] ⊢$_{ℕat}$P⟦x⟧) ⇨ ⊢$_{ℕat}$ ∀[x:t] P⟦x⟧

Proof. Suppose to obtain a contradiction that ¬∀[x:t]P⟦x⟧. Let x$_0$:t such that ¬P⟦x$_0$⟧. Then ⊢$_{ℕat}$P⟦x$_0$⟧ by the hypothesis of the theorem. P⟦x$_0$⟧ follows by inferential soundness, which is a contradiction.

ℕat is algorithmically inexhaustible

That all the theorems of a theory can be obtained by computationally enumerating them from axioms has long been a default assumption of philosophers of logic. However, the theory ℕat violates this assumption because there are uncountable instances of the induction axiom thereby raising questions about the theory ℕat↾*String*, which has the following induction axiom, **which has countable instances** because strings are countable**.**:

∀[i:N_+, P:(*PropositionOfOrder*$_{NAT}$◁i▷^{N})↾*String*]
(P⟦0⟧ ∧ ∀[i:N] P⟦i⟧⇨P⟦+$_1$[i]⟧) ⇨ ∀[i:N] P⟦i⟧

Definitions.

- *Total*◁t_1, t_2▷ ≡ (t_1→t_2)∋λ[f] ∀[x:t_1] ∃[y:t_2] f$_■$[x]=y
- *Onto*◁t_1, t_2▷ ≡ (t_1→t_2)∋λ[f] ∀[y:t_2] ∃[x:t_1] f$_■$[x]=y
- *ProvedTotal*$_{ℕat↾String}$◁t_1, t_2▷ ≡
  (t_1→t_2)↾*String*∋λ[f]⊢$_{ℕat↾String}$ f:*Total*◁t_1,t_2▷

Theorem. *Theorem*◁ℕat↾*String*▷ is computationally enumerable, i.e., there is a procedure Theorems:*ProvedTotal*$_{ℕat↾String}$ such that Theorems:*Onto*◁[N], *Theorem*◁ℕat↾*String*▷▷

Corollary. *ProvedTotal*$_{ℕat↾String}$ is computationally enumerable, i.e., there is a procedure ProvedTotals:*ProvedTotal*$_{ℕat↾String}$ such that ProvedTotals:*Onto*◁[N], *ProvedTotal*$_{ℕat↾String}$▷

Definition. Define the procedure Diagonal as follows:

Diagonal$_■$[i:N] ≡ 1+(ProvedTotals$_■$[i])$_■$[i]

Lemma. Diagonal:*ProvedTotal*$_{ℕat↾String}$

Proof. Suppose i:N. Let f:*ProvedTotal*$_{ℕat↾String}$=ProvedTotals$_■$[i] and let j:N=f$_■$[i]. Therefore Diagonal$_■$[i]=1+j. Consequently, ⊢$_{ℕat↾String}$ Diagonal:*Total*◁[N], N▷.

Lemma. ¬Diagonal:*ProvedTotal*$_{ℕat↾String}$

Proof. Diagonal differs from every procedure enumerated by ProvedTotals.

Theorem. ℕat↾*String* is inconsistent [Church 1934], i.e.,

∃[Ψ:*Proposition*◁ℕat↾*String*▷] ⊢$_{ℕat}$Ψ∧¬Ψ

Proof. Let Ψ=Diagonal:*ProvedTotal*$_{ℕat↾String}$

According to [Church 1934], the above theorem means that "there is no sound basis for supposing that there is such a thing as logic." **Contrary to [Church 1934], the conclusion in this article is to abandon the assumption that theorems of a theory must be computationally enumerable while retaining the requirement that proof checking must be computationally decidable.**

ℕat is algorithmically inexhaustible, i.e., nonalgorithmic creativity will be forever required to develop new ℕat axioms abstracted from strings thereby reinforcing the intuition behind [Franzén, 2004]

V. Actor Model

[Church 1932] and [Turing 1936] developed a model of computation time based on the concept of an *algorithm*, which by definition is provided an input from which it is to compute a value without external interaction. **After physical computers were constructed, they soon diverged from computing only algorithms meaning that the Church/Turing theory of computation no longer applied to computation in practice because computer systems are highly interactive as they compute.** Actors [Hewitt, et. al 1973] (axiomatized in this article) remedied the omission to provide for scalable computation. An Actor machine can be millions of times faster than any corresponding pure Logic Program or parallel nondeterministic λ expression. Since the time of this early work, Actors have grown to be one of the most important paradigms in computing [Hewitt and Woods 2019; Milner 1993].

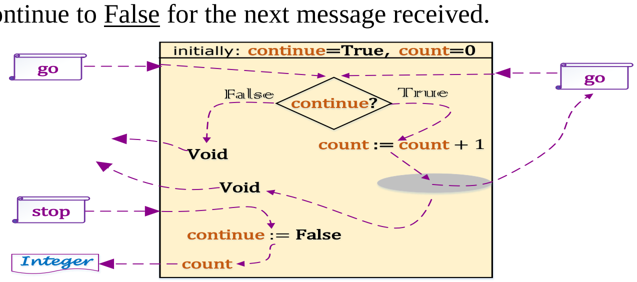
Of course, earlier work made huge pioneering contributions: λ expressions [Church 1932] play an important role in programming languages. Turing Machines [Turing 1936] inspired development of the stored program sequential computer and Logic Programs are fundamental to Scalable Intelligent Systems. [Hewitt 2019]

Computation that cannot be done by λ Calculus, Nondeterministic Turing Machines, or pure Logic Programs

Actor machines can perform computations that a no λ expression, nondeterministic Turing Machine or pure Logic Program can implement. Below is an example of a very simple computation that cannot be performed by a nondeterministic Turing Machine:

There is an *always-halting* Actor machine that can compute an integer of unbounded size. This is accomplished using variables count initially 0 and continue initially <u>True</u>. The computation is begun by concurrently sending two messages to the Actor machine: a stop request that will return an integer n formalized as Output[n] and a go message that will return Void. The Actor machine operates as follows:

- When a go message is received: If continue is <u>True</u>, increment count by 1, send this Actor machine a go message in a hole of the region of mutual exclusion, and afterward return Void. If continue is <u>False</u>, return Void.
- When a stop message is received, return count and set continue to <u>False</u> for the next message received.

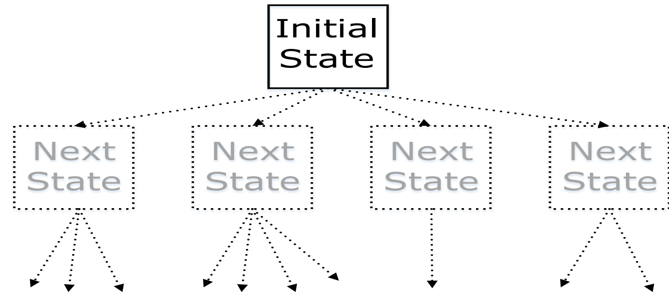


Resend **go** message until **stop** message received

Theorem. There is no λ expression, nondeterministic Turing Machine, or pure Logic Program that implements the above computation.

Proof [Plotkin 1976]:

> "Now the set of initial segments of execution sequences of a given nondeterministic program P, starting from a given state, will form a tree. The branching points will correspond to the choice points in the program. Since there are always only finitely many alternatives at each choice point, the branching factor of the tree is always finite. That is, the tree is finitary. Now König's lemma says that if every branch of a finitary tree is finite, then so is the tree itself. In the present case this means that if every execution sequence of P terminates, then there are only finitely many execution sequences. So if an output set of P is infinite, it must contain a nonterminating computation."



Nondeterministic Choice

Limitations of 1st-order Logic for Concurrent Computation

Theorem. It is well known that there is no 1st-order theory for the above Actor machine.

Proof. Every 1st-order theory is compact meaning that every inconsistent set of propositions has a finite inconsistent subset. Consequently, to show that there is no 1st-order theory, it is sufficient to show that there is an inconsistent set of propositions such that every finite subset is consistent. Let Output[i] mean that i is output, The set of propositions {¬Output[i] | i:N} is inconsistent but every *finite* subset S is consistent because the Actor machine output might be larger than any output in S.

Actors have fundamentally transformed the foundations and practice of computation since the initial conceptions of Turing and Church. Although 1st-order propositions can be useful (e.g. in testing 1st-order propositions for satisfiability), message passing illustrates why 1st-order logic cannot be the foundation for theories in Computer Science.

Actors in Practice

An interface can be defined using an interface name, "<u>interface</u>", and a list of message handler signatures, where message handler signature consists of a message name followed by argument types delimited by "**[**" and "**]**", "**→**", and a return type. For example, the interface type *ReadersWriter* can be defined as follows:

ReadersWriter <u>interface</u>
read[*Query*]→ *ReadResponse*,
write[*Update*]→ *WriteResponse*

Holes in regions of mutual exclusion

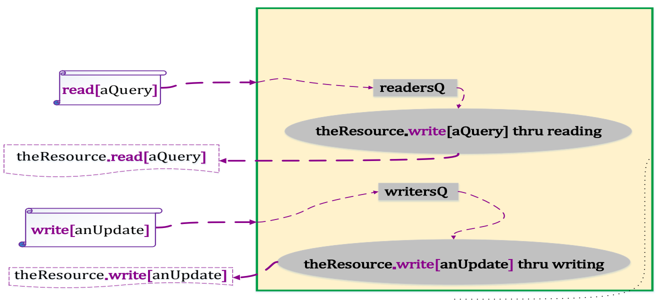
Holes in regions of mutual exclusion (Swiss cheese) [Hewitt and Atkinson 1979; Atkinson 1980] is a generalization of mutual exclusion with the following goals:

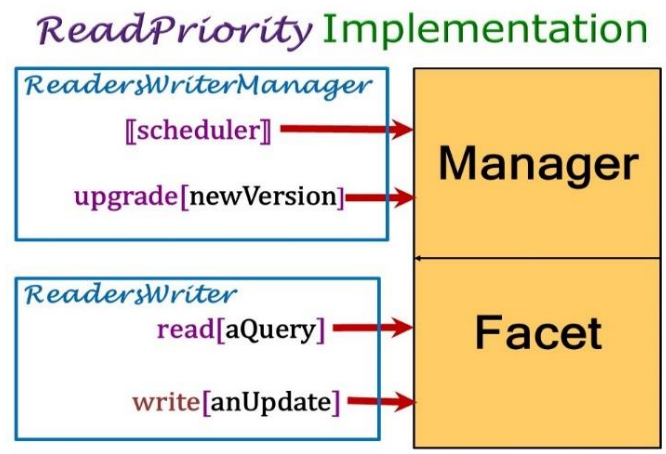
- *Generality:* Conveniently program any scheduling policy
- *Performance:* Support maximum performance in implementation, e.g., the ability to minimize locking and to avoid repeatedly recalculating a condition for proceeding.
- *Understandability:* Invariants for the variables of a mutable Actor should hold whenever entering or leaving the region of mutual exclusion.
- *Modularity:* Resources requiring scheduling should be encapsulated so that it is impossible to use them incorrectly.

Coordinating activities of readers and writers in a shared resource is a classic problem. The fundamental constraint is that multiple writers are not allowed to operate concurrently and a writer is not allowed to operate concurrently with a reader.

Below is a read priority implementation of a readers/writer scheduler for a database in which it is forbidden for a writer to operate concurrently with any other activity (cf. [Hoare 1974; Brinch Hansen 1996]):

```
ReadPriority⟦aDatabase:ReadersWriter⟧ ↦
  // Invariant: Nonempty⟦writing⟧ ⇨ IsEmpty⟦reading⟧
  #Local(#FIFO(writersQ, readersQ),
                                   // queues of suspended activities
         #Crowd(reading),          // crowd of active reading
         #AtMostOne(writing)),  // at most one writing
  #Handler(getScheduler ↦ As myScheduler,
           upgrade[newVersion] ↦
             #CancelAll(readersQ, writersQ, reading, writing)
               for Become newVersion)
  myScheduler implements ReadersWriter #Handler(
    read[aQuery] ↦
      Enqueue readersQ
             when #SomeNonempty(writing, writersQ, readersQ)
        for                    // Require: #IsEmpty writing
         Permit readersQ
           for  aDatabase.read[aQuery] thru reading
               afterward        // Require: #IsEmpty writing
                 permit writersQ when IsEmpty reading
                     else readersQ when
                         #AllEmpty(writing, writersQ)
    write[anUpdate] ↦
      Enqueue writersQ when #SomeNonempty(reading,
                             readersQ, writing, writersQ)
        for              // Require: #IsEmpty(writing, reading)
         aDatabase.write[anUpdate] thru writing
           afterward    // Require: #AllEmpty(writing, reading)
              Permit readersQ else writersQ)
```





Note:

1. At most one activity is allowed to execute in the region of mutual exclusion of *ReadPriority*.
2. The region of mutual of exclusion has holes illustrating that an Actor is **not** a sequential process (thread) in which control moves sequentially through a program.
3. An invariant holds when an Actor is created and when entering/leaving a continuous section of a region of mutual exclusion.
4. An implementation, e.g. *ReadPriority*, differs from a class [Dahl and Nygaard 1967] as follows:
   - An implementation can use **multiple** other implementations using namespaces to prevent ambiguity [cf. ISO 2017].
   - An implementation **cannot** be subclassed [Dahl and Nygaard 1967] in order to prevent impersonation by other types.

A *ReadPriority* implementation has the following invariant:

Nonempty⟦writing⟧ ⇨ IsEmpty⟦reading⟧

which holds because of Actor Induction as follows [Turing 1949, Hewitt 2017-2019]:

- The invariant holds when a *ReadPriority* implementation is created.
- If the invariant holds in a *ReadPriority* implementation when a communication is received, then it holds when leaving.

Starvation of activities suspended in readersQ and writersQ as is prevented in a *ReadPriority* implementation as follows:

- An activity in readersQ progresses when
  1. A read to the database is started by another activity
  2. If writersQ and writing are both empty after the read to the database is completed.by another activity
  3. Else after the next write to the database is finished.
- An activity in writersQ progresses when
  1. If readersQ is empty when a write to the database is completed.by another activity
  2. Else when reading becomes smaller when reading the database is completed by another activity.

Reading throughput is maintained by permitting readersQ when another activity starts a read to the database.

Categorical Axiomatization of Actors

Let x[e] be the behavior of Actor x at local event e, Com be the type for a communication, and Behavior be the type for a procedure that maps a communication received to an outcome that has a finite set of created Actors, a finite set of sent communications, and a behavior for the next communication received.

ActorTheory categorically axiomatises Actors using the following axioms where ↷ (read as "precedes") is transitive and irreflexive and Info[x] is the information in the Actor addresses of x:

- Primitive Actors
  - ∀[i:*N*] i:*Actor* // natural numbers are Actors
  - ∀[x_1,x_2:*Actor*] [x_1, x_2]:*Actor* // a 2-tuple of Actors is an Actor
- An Actor's event ordering
  - ∀[x:*Actor*, c:*Com*] Initial_x↷Received_x[c]↷After_x[c]
  - ∀[x:*Actor*, c_1,c_2:*Com*] $c_1 \neq c_2$ ⇒ $\text{Received}_x[c_1]$↷$\text{Received}_x[c_2]$ ∨ $\text{Received}_x[c_2]$↷$\text{Received}_x[c_1]$
  - ∀[x:*Actor*, c_1:*Com*] ∄[c_2:*Com*] $\text{Received}_x[c_1]$↷$\text{Received}_x[c_2]$↷$\text{After}_x[c_1]$
- An Actor's behavior change
  - ∀[x:*Actor*, c_1:*Com*] (∄[c_2:*Com*] $\text{Received}_x[c_2]$↷$\text{Received}_x[c_1]$) ⇒ x[$\text{Received}_x[c_1]$]=x[Initial_x]
  - ∀[x:*Actor*, c:*Com*] x[After_x[c]] = (x[Before_x[c]])[c]
  - ∀[x:*Actor*, c:*Com*]
    Let processing = Info[Before_x[c]]⊔Info[Received_x[c]] ⊔Info[Created_x[c]]
    // processing is information about addresses
    // before c was received, received in c,
    // and created processing c
    in Info[After_x[c]]⊑processing
    ∧ Info[Sent_x[c]]⊑processing
- Discreteness of Actors event ordering
  ∀[e_1, e_2:*Event*] Finite⟦{e:*Event*| e_1↷e↷e_2}⟧
  // There are only finitely many events
  // in ↷ between two events.
- Actor Induction
  ∀[x:*Actor*, P predicateOn *Behavior*]
  (P⟦x[Initial_x]⟧ ∧ ∀[c:*Com*] P⟦x[Received_x[c]]⟧⇨P⟦x[After_x[c]]⟧)
  ⇨ ∀[c:*Com*] P⟦x[Received_x[c]]⟧ ∧ P⟦x[After_x[c]]⟧

Note that the above axioms do not require that every communication sent must be received. However, ActorScript [Hewitt and Woods assisted by Spurr 2015] provides that every request will either throw a *TooLong* exception or respond with the response sent to its customer.

Theorem. Unique Categoricity of ActorTheory, i.e., if *M* is a type satisfying the axioms for ActorTheory, then there is a unique isomorphism between *M* and *Actor*.

Thesis: **Any digital system can be directly modeled and implemented using Actors.**

In many practical applications, the parallel λ-calculus and pure Logic Programs can be thousands of times slower than Actor implementations.

VI. Mathematical Theories of Computer Science

Standard Mathematical Theories of Computer Science

Although theorems of mathematical theories in higher order logic are not computationally enumerable, proof checking is computationally decidable. Strong types can be used categorically axiomatize [Hewitt 2017-2019] up to a unique isomorphism a mathematical theory 𝕋 for the model *M* for each of the following: Natural Numbers, Real Numbers, Ordinals, Computable Procedures, and Actors with the following properties:

- 𝕋 is categorical for *M*, i.e., if *X* satisfies the axioms of 𝕋, then is *X* isomorphic to *M* by a unique isomorphism.
- 𝕋 is **not** compact
- 𝕋 has Instance Adequacy, i.e.,
  ∀[*t*:*TypeIn*◁𝕋▷ P predicateOn$_{\mathbb{T}}$ *t*]
  (∀[x:*t*] $\vdash_{\mathbb{T}}$P⟦x⟧) ⇨ $\vdash_{\mathbb{T}}$ ∀[x:*t*] P⟦x⟧
- $\vdash^{p}_{\mathbb{T}}$ Ψ is computationally decidable for Ψ:*Propositon*◁𝕋▷ and p:*ProofChecker*◁𝕋▷

Information Invariance

Information Invariance is a fundamental technical goal of logic consisting of the following:

1. *Soundness of inference*: information is not increased by inference
2. *Completeness of inference*: all valid inferences are included.

Criteria for Mathematical Foundations

Computer Science brought different concerns and a new perspective to mathematical foundations including the following requirements (building on [Maddy 2018]):

- *Practicality* is providing powerful machinery so that arguments (proofs) can be short and understandable and
- *Generality* is formalizing inference so that all of mathematics can take place side-by-side. Strong types provide generality by formalizing theories of the natural numbers, reals, ordinals, set theory, groups, lambda calculus, and Actors side-by-side.
- *Shared Standard* of what counts as legitimate mathematics so people can join forces and develop common techniques and technology. According to [Burgess 2015]:

  "To guarantee that rigor is not compromised in the process of transferring material from one branch of mathematics to another, it is essential that the starting points of the branches being connected ... be compatible. ... The only obvious way ensure compatibility of the starting points ... is ultimate to derive all branches from a common unified starting point."

This article describes such a common unified starting point including natural numbers, reals, ordinals, sets, groups, geometry, algebra, lambda calculus, and Actors that are axiomatized up to a unique isomorphism.

- *Abstraction* so that fundamental mathematical structures can be characterized up to a unique isomorphism including natural numbers, reals, ordinals, sets, groups, lambda calculus, and Actors.
- *Guidance* is for practioners in their day-to-day work by providing relevant structures and methods free of extraneous factors. This article provides guidance by providing strong parameterized types and intuitive categorical inductive axiomatizations of natural numbers, ordinals, sets, lambda calculus, and Actors.
- *Meta-Mathematics* is the formalization of logic and rules of inference. The mathematical theories described in this article facilitate meta-mathematics because inference is directly on propositions without having to be coded as integers as in [Gödel 1931].
- *Automation* is facilitated in this article by making type checking very easy and intuitive along as well as incorporating Jaśkowski natural deduction for building an inferential system that can be used in everyday work.
- *Risk Assessment* is the danger of contradictions emerging in classical mathematical theories. This article formalizes long-established and well-tested mathematical practice while blocking all known paradoxes. (See appendix on paradoxes.) Confidence in the consistency of Nat and ActorTheory is based on the way that they are inductively constructed bottom-up.
- *Monsters* [Lakatos 1976] are unwanted elements in models of classical mathematical theories. ActorTheory precisely characterizes what is digitally computable leaving no room for "monsters" in models. Having a unique model up to isomorphism in classical mathematical theories is crucial for cybersecurity.

Intuitive categorical *inductive* axiomatizations of natural numbers, propositions, types, ordinals, sets, lambda calculus, and Actors promote confidence in operational consistency.

Consistent mathematical theories can be freely used in (inconsistent) empirical theories without introducing additional inconsistency.

VII. Cybersecurity Crisis

The current disastrous state of cybersecurity cries out for a paradigm shift away from 1st-order logic (the basis of Gödel's results discussed in this article) as the foundation for mathematical theories of Computer Science because of the following deficiencies:

- unwanted monsters in models of theories
- inconsistencies in theories caused by compactness
- inconsistencies in theories that can formalize their own provability
- being able to infer each and every proposition (including nonsense) from an inconsistency in an empirical theory even though it may not be apparant that the theory is inconsistent.

Thus Computer Science must move beyond the consensus claimed by [G. H Moore 1988] as follows: "To most mathematical logicians working in the 1980s, first-order logic is the proper and natural framework for mathematics."

According to [Kuhn 2012],

> "The decision to reject one paradigm is always simultaneously the decision to accept another. First, the new candidate must seem to resolve some outstanding and generally recognized problem that can be met in no other way. Second, the new paradigm must promise to preserve a relatively large part of the concrete problem solving activity that has accrued to science through its predecessor ...
>
> At the start, a new candidate for paradigm shift may have few supporters, and on occasions supporters' motives may be suspect. Nevertheless, if they are competent, they will improve it, explore its possibilities, and show what it would be like to belong to the community guided by it. And as that goes on, if the paradigm is one destined to win its fight, the number and strength of the persuasive arguments in its favor will increase. More scientists will then be converted, the exploration of the new paradigm will go on. Gradually, the number of experiments, instruments, and books upon the paradigm will multiply...
>
> Though a generation is sometimes required to effect the shift, scientific communities have again and again been converted to new paradigms. Furthermore, these conversions occur not despite the fact that scientists are human but because they are. ... Conversions will occur a few at a time until, after the last holdouts have died, the whole profession will again be practicing under a single, but now different paradigm."

The necessity to give up a long-held intuitive assumption has often held back the development of a paradigm shift. For example, the Newtonian assumption of absolute space-time had to be given up in the theory of relativity. Also, physical determinacy had to be abandoned in quantum theory. According to [Church 1934]:

> "Indeed, if there is no formalization of logic as a whole [i.e. theorems are not computationally enumerable], then there is no exact description of what logic is, for it in the very nature of an exact description that it implies a formalization. And if there no exact description of logic, then there is no sound basis for supposing that there is such a thing as logic."

Contrary to [Church 1934], the conclusion in this article is to abandon the assumption that theorems of a theory must be computationally enumerable while retaining the requirement that proof checking must be computationally decidable.

The Establishment has made numerous mistakes during paradigm shifts. For example, Arthur Erich Has derived the radius of the ground state of the hydrogen atom [Haas 1910], anticipating Niels Bohr work by 3 years. Yet in 1910 Haas's article was rejected and his ideas were termed a "carnival joke" by Viennese physicists. [Hermann 2008] On the other hand, Enrico Fermi received the 1938 Nobel prize for the discovery of the nonexistent elements "Ausonium" and "Hesperium", which were actually mixtures of barium, krypton and other elements. [Fermi 1938]

How the Computer Science cybersecurity crisis will proceed is indeterminate, including the following possibilities.

- muddle along without fundamental change
- shift to something along the lines proposed in this article
- shift to some other proposal that has not yet been devised

Cybersecurity issues can provide focus and direction for fundamental research in Computer Science.

VII. Related Work

Recent work has centered on constructive type theory which has type $t_1 \rightarrow t_2$, which is the type of *computable* procedures t_1 into t_2, but does *not* have ${t_2}^{t_1}$, which is the type of *all* functions from t_1 into t_2. Also, constructive type theory relies on *Propositon*◁𝕋▷=*Theorem*◁𝕋▷ with the unfortunate consequence that type checking is *computationally undecidable* and it is difficult to reason about unprovable propositions.

Extensions of Isabelle/HOL [Gordon 2016] seem more suitable for formalizing classical mathematics than constructive type theory.

VIII. Conclusion

This article strengthens the position of working mathematicians as follows:

- Providing usable theories of standard mathematical theories of computer science (e.g. Natural Numbers and Actors) such that there is only one model up to a unique isomorphism. The approach in this article is to embrace **all** of the most powerful tools of classical mathematics in order to provide mathematical foundations for Computer Science. Fortunately, these foundations are technically simple so they can be readily automated, which will enable improved collaboration between humans and computer systems.
- Allowing theories to freely reason about theories
- Providing a theory that precisely characterizes all digital computation as well as a strongly-typed programming language that can directly, efficiently, and securely implement every Actor computation.
- Providing in foundation for well-defined classical theories of natural numbers and Actors for use in reasoning by theories of practice in Scalable Intelligent Systems that are (of necessity) pervasively inconsistent.

Blocking known paradoxes makes classical mathematical theories safer for use in Scalable Intelligent Systems by preventing security holes. **Consistent strong mathematical theories can be freely used without introducing additional inconsistent information into inconsistency robust empirical theories that will be the core of future Intelligent Applications**.

Inconsistency Robustness [Hewitt and Woods assisted by Spurr 2015] is performance of information systems (including scientific communities) with massive pervasively-inconsistent information. Inconsistency Robustness of the community of professional mathematicians is their performance repeatedly repairing contradictions over the centuries. In the Inconsistency Robustness paradigm, deriving contradictions has been a progressive development and not "game stoppers." Contradictions can be helpful instead of being something to be "swept under the rug" by denying their existence, which has been repeatedly attempted by dogmatic theoreticians (beginning with some Pythagoreans). Such denial has delayed mathematical development.

For reasons of computer security, Computer Science must abandon the thesis that theorems of fundamental mathematical theories must be computationally enumerable. This can be accomplished while preserving almost all previous mathematical work except the 1st-Order Thesis [Barwise 1985]. **Automation of the proofs in this article is within reach of the state of the art which will enable better collaboration between humans and computer systems.**

Having a powerful system is important because computers must be able to formalize all logical inferences (including inferences about their own inference processes) so that computer systems can better collaborate with humans.

Acknowledgment

Extensive conversations with Dan Flickinger, Fanya Montalvo, John Perry, Gordon Plotkin, Shankar, and John Woods were extremely helpful in developing ideas in this article. Richard Waldinger made very helpful comments. Shankar and David Israel pointed out that the article needed to be more explicit on the relationship of Wittgenstein's proof to [Gödel 1931]. Kevin Hammond suggested including the section on related work. Dan Flickinger suggested improvements in the section on paradigm shifts.

Appendix: Mathematical Notation

Notation for mathematical propositions, mathematical terms, and procedural expressions is formalized in this appendix.

Mathematical *Proposition* is a discrimination of the following patterns:

- ¬Ψ1, Ψ1∧Ψ2:*PropositionOfOrder*◁i▷ where Ψ1,Ψ2:*PropositionOfOrder*◁i▷ and i:N_+
- (x1=x2):*PropositionOfOrder*◁1▷ where x1,x2:*Term*◁*t*▷ and *t* is a type
- (x:*t*):*PropositionOfOrder*◁1▷ where *t* is a type
- P⟦x⟧:*PropositionOfOrder*◁i+1▷ where x:*Term*◁*t*▷, *t* is a type and P:*Term*◁*Proposition*◁i▷^{t}▷ and i:N_+
- (Ψ1⊢Ψ2):*PropositionOfOrder*◁i▷ where i:N_+ and Ψ1,Ψ2:*PropositionOfOrder*◁i▷
- (Ψ1 $\vdash^{p}$ Ψ2):*PropositionOfOrder*◁i▷ where p:*Term*◁*ProofChecker*▷, and Ψ1,Ψ2:*PropositionOfOrder*◁i▷ and i:N_+
- (⊩$_t$Ψ):*PropositionOfOrder*◁i▷ where *t* is a type, Ψ:*PropositionOfOrder*◁i▷ and i:N_+
- ⌊s⌋:*PropositionOfOrder*◁i▷ (abstraction of s) where s:*PropositionOfOrder*◁i▷↾*String* with no

free variables, and i:N_+

Mathematical *Proposition from String* is a discrimination of the following patterns:

- “"¬" s_1”,“s_1 "∧" s_2”:*PropositionofOrder*◁i▷↾*String* where s_1,s_2:*PropositionofOrder*◁i▷↾*String* and i:N_+
- “x_1"="x_2”: *PropositionofOrder*◁1▷↾*String* where where x_1,x_2:*Term*↾*String*
- “x_1 ":" x_2”: *PropositionofOrder*◁1▷↾*String* where x_1,x_2:*Term*↾*String*
- “"∀[" x ":" *t* "]" s”:*PropositionofOrder*◁i+1▷↾*String* where *t* is a type, x:*Variable*◁*t*▷↾*String* and s:*PropositionofOrder*◁i▷↾*String*
- “P "⟦" x "⟧"”: *PropositionofOrder*◁i+1▷↾*String* where x:*Term*◁*t*▷↾*String*, i:N_+ and P:*Term*◁*PropositionofOrder*◁i▷^{t}↾*String*▷
- “s_1"⊢"s_2”: *PropositionofOrder*◁I+1▷↾*String* where s_1,s_2:*PropositionofOrder*◁i▷↾*String*, and i:N_+.
- “s_1 " ⊢"$\underline{p}$ s_2”:*PropositionofOrder*◁i+1▷↾*String* where p:*Term*◁*ProofChecker*▷↾*String*, s_1,s_2:*PropositionofOrder*◁i▷↾*String*,, and i:N_+
- “"⊫"$_t$s”:*PropositionofOrder*◁i+1▷↾*String* where s:*PropositionofOrder*◁i▷↾*String*, i:N_+ and *t*:*Term*↾*String*, .
- “"⌊" s "⌋"”:*PropositionofOrder*◁i▷↾*String* is abstraction of s where s:*PropositionofOrder*◁i▷↾*String* with no free variables, and i:N_+

Mathematical *Term* is a discrimination of the following patterns:

- *Boolean*:*Constant*◁*TypeOf*◁*Boolean*▷▷, *N*:*Constant*◁*TypeOf*◁*N*▷▷, and *Actor*:*Constant*◁*TypeOf*◁*Actor*▷▷
- x:*Term*◁*t*▷ where x:*Constant*◁*t*▷ and *t* is a type
- x:*Term*◁*t*▷ where x:*Variable*◁*t*▷ and *t* is a type
- [x_1, x_2]:*Term*◁[t_1, t_2]▷ where x_1:*Term*◁t_1▷, x_2:*Term*◁t_2▷, and t_1 and t_2 are types
- (x_1 <u>if</u> <u>True</u> <u>then</u> x_2 , <u>False</u> <u>then</u> x_3):*Term*◁*t*▷ where x_1:*Term*◁*Boolean*▷, x_2,x_3:*Term*◁*t*▷ and *t* is a type
- (λ[x:t_1] y):*Term*◁$t_2^{t_1}$▷ where x:*Variable*◁t_1▷, y:*Term*◁t_2▷ and t_1 and t_2 are types
- f[x]:*Term*◁t_2▷ where f:*Term*◁$t_2^{t_1}$▷, x:*Term*◁t_1▷, and t_1 and t_2 are types
- ⌊x⌋:*Term*◁*t*▷ is abstraction of x where x:*Term*◁*t*▷↾*String* and *t* is a type

Procedural *Expression* is a discrimination of the following:

- x:*Expression*◁*t*▷ where x:*Constant*◁*t*▷ and *t* is a type
- x:*Expression*◁*t*▷ where x:*Identifier*◁*t*▷ and *t* is a type
- [e_1, e_2]:*Expression*◁[t_1, t_2]▷ where e_1:*Expression*◁t_1▷, e_2:*Expression*◁t_2▷, and t_1 and t_2 are types
- (e_1 <u>if</u> <u>True</u> <u>then</u> e_2 , <u>False</u> <u>then</u> e_3):*Expression*◁*t*▷ where e_1:*Expression*◁*Boolean*▷, e_2,e_3:*Expression*◁*t*▷ and *t* is a type
- (λ[x:t_1] y):*Expression*◁t_1→t_2▷ where x:*Identifier*◁t_1▷, y:*Expression*◁t_2▷ and t_1 and t_2 are types
- x.m:*Expression*◁t_2▷ where m:*Expression*◁t_1▷, x is an Actor with a message handler with signature of type *Expression*◁t_1→t_2▷, and t_1 and t_2 are types
- *I*⟦x_1, ..., x_n⟧:*Expression*◁*I*▷ where *I* is an Actor implementation and x_1, ..., x_n are expressions.
- ⌊x⌋:*Expression*◁*t*▷ is abstraction of x where x:*Expression*◁*t*▷↾*String* and *t* is a type

APPENDIX: MATHEMATICAL PARADOXES

Inconsistencies in fundamental mathematical theories of Computer Science are dangerous because they can be used to create security vulnerabilities. Strong types are extremely important because they block *all* known paradoxes including the ones in this appendix.

Russell [Russell 1902]

- Russell’s paradox for sets is resolved as follows: the type of all sets restricted to ones that are not elements of themselves is just the type of all sets because **no** set is an element of itself.
- Russell’s paradox for predicates is resolved as follows: The mapping P↦¬P⟦P⟧ has **no** fixed point because ¬P⟦P⟧ has order one greater than the order of P because P is a predicate variable.

Gödel [Gödel 1931]

Gödel ’s Paradox is blocked because the mapping Ψ↦⊬Ψ does **not** have a fixed point because the order of ⊬Ψ is greater than the order of Ψ since Ψ is a propositional variable.

Curry [Curry 1941]

Curry’s Paradox is blocked because the mapping p↦p⇒Ψ does **not** have a fixed point because the order of p⇒Ψ is greater than the order of p since p is a propositional variable.

Löb [Löb 1955]

Löb’s Paradox is blocked because the mapping p↦((⊢p)⇒Ψ) does **not** have a fixed point because the order of (⊢p)⇒Ψ is greater than the order of p since p is a propositional variable.

Yablo [Yablo 1985]
Yablo's Paradox is blocked because the mapping P↦λ[i:*N*]∀[j:*N*] j>i⇨¬P⟦j⟧ does **not** have a fixed point because the order of λ[i:*N*] ∀[j:*N*]j>i⇨¬P⟦j⟧ is greater than the order of P since P is a predicate variable [cf. Priest 1997].

Berry [Russell 1906]
Berry's Paradox can be formalized using the proposition Characterize◁i▷⟦s, k⟧ meaning that the string s characterizes the integer k as follows where i:N_+:

- *Berry*◁i▷≡(*Term*◁*PropositionofOrder*◁i▷^{N})↾*String*
- Characterize◁i▷⟦s:*Berry*◁i▷, k:*N*⟧ ≡ ∀[x:*N*]⌊s⌋⟦x⟧⇔ x=k

The Berry Paradox is to construct a string for the proposition that holds for integer n if and only if every string with length less than 100 does not characterize n using the following definition:

BerryString:*Berry*◁i+1▷ ≡
"λ[n:*N*] ∀[s:*PropositionOfOrder*◁i▷↾*String*]
Length[s]<100 ⇨ ¬Characterize◁i▷⟦s, n⟧"

Note that

- Length[BerryString]<100.
- *Berry*◁i▷∋λ[s] Length[s]<100 is finite.
- Therefore, *BerryNumber* is finite where
BerryNumber ≡
N_+∋λ[i] ∃[s:*Berry* ◁i▷]
Length[s]<100 ∧ Characterize◁i▷⟦s, i⟧
- ∃[i:N_+] i:*BerryNumber* because is N_+ is infinite.
- LeastBerry ≡ Least[*BerryNumber*]
- ⌊BerryString⌋⟦LeastBerry⟧ =
∀[s:*Berry*◁i▷]
Length[s]<100 ⇨ ¬Characterize◁i▷⟦s, LeastBerry ⟧

However BerryString:*Berry*◁i+1▷ **cannot be substituted** for s:*Berry*◁i▷. Consequently, **the Berry Paradox as follows does not hold**:

⌊BerryString⌋⟦LeastBerry⟧
⇔ ¬Characterize◁i▷⟦BerryString, LeastBerry⟧

APPENDIX: ORDINALS

Ordinals (denoted by type *O*) can be axiomatized up to a unique isomorphism in a theory called 𝕆𝕣𝕕 [Hewitt 2016-2019].

Theorem. 𝕆𝕣𝕕 is inferentially undecidable, i.e.,
∃[Ψ:*Proposition*◁𝕆𝕣𝕕▷] (⊬$_{\mathbb{Ord}}$Ψ) ∧ (⊬$_{\mathbb{Ord}}$ ¬Ψ)

Proof Suppose to obtain a contradiction that 𝕆𝕣𝕕 is inferentially decidable and consequently
∀[x:*Expression*◁*N*▷] (⊢$_{\mathbb{Ord}}$ Halt[x]) ∨ (⊢$_{\mathbb{Ord}}$¬Halt[x])
where Halt[x] means that expression x halts. Since only countably many instances of the Ordinal induction axiom could have been used in the above proofs, the halting problem is computationally decidable by computationally enumerating the proofs, which is a contradiction.

Thesis. 𝕆𝕣𝕕 is inferentially complete, i.e., all valid mathematical inference for the Ordinals can be carried out in 𝕆𝕣𝕕. (The thesis should be formalized as a mathematical theorem).

Let Ω be the least Ordinal of cardinality *Boolean*^{N}.

Lemma. ⊢$_{\mathbb{Ord}}$ ∀[α:*O*] α<Ω ⇨ Countable[α]

Proof. Immediate from the definition of Ω.

Theorem. Continuum Hypothesis for Ordinals, i.e.,
⊢$_{\mathbb{Ord}}$ ∄[α:*O*] |ω|<|α|<|Ω|, where ω is the least Ordinal of cardinality *N* and |δ| is the cardinality of Ordinal δ.

Proof. Follows immediately from the above lemma.

Consequently, the **opposite** of the 1st-order Gödel/Cohen [Cohen 1963-1964] result holds for Ordinals.

The Continuum Hypothesis for Ordinals is the most important version of the Continuum Hypothesis for Computer Science. The version of the Continuum Hypothesis for untyped sets [cf. Kreisel 1967, page 152; Feferman 2011] is less important because theories in Computer Science use a mixture of lists (e.g. *List*◁*N* ▷), sets (e.g. *Set*◁*List*◁*N*▷▷), trees, etc. parameterized by types instead of the ill-defined notion of an untyped set of sets. To defeat cyberattacks, 𝕊𝕖𝕥𝕋𝕙𝕖𝕠𝕣𝕪 must be axiomatized up to a unique isomorphism. [cf. Hewitt 2017-2019]

REFERENCES

S. Artemov. *The Provability of Consistency* ArXiv. March 18, 2019.
J. Avigad, G. Ebner, and S. Ullrich. *The Lean Reference Manual: Release 3.3.0*. September 6. 2018.
R. Atkinson. *Automatic Verification of Serializers* MIT Doctoral Dissertation. June, 1980.
S. Awodey and E. Reck. *Completeness and Categoricity. Parts I and II: Nineteenth-century Axiomatics to Twentieth-century Metalogic*. History and Philosophy of Logic. Vol. 23. 2002.
J. Barwise. *Model-Theoretic Logics: Background and Aims* Model Theoretic Logics. Springer-Verlag. 1985.
C. Benzmüller, N. Sultana, L. Paulson and F. Theiß. *The Higher-Order Prover Leo-II* Journal of Automated Reasoning. Vol. 55. Issue 4. December 2015.
P. Brinch Hansen. *Monitors and Concurrent Pascal: A Personal History* SIGPLAN Notices. March 1993.
C. Burali-Forti. *Una questione sui numeri transfiniti* Rendiconti del Circolo Matematico di Palermo. 1897
J. Burgess. *Rigor and Structure* Oxford University Press. 2015.
A. Church. *A set of postulates for the foundation of logic* Annals of Mathematics. Series 2. 33 (2). 1932.
A. Church. *The Richard Paradox*. Proceedings of American Mathematical Society. Vol. 41. No. 6. 1934.
P. Cohen. *The Independence of the Continuum Hypothesis* US National Academy of Sciences. 1963-1964.
T. Coquand and G. Huet. *The calculus of constructions.* Technical Report 530, INRIA, Centre de Rocquencourt, 1986.
H. Curry. *Some Aspects of the Problem of Mathematical Rigor* Bulletin of the American Mathematical Society Vol. 4. 1941.
O. Dahl and K. Nygaard. *Class and subclass declarations* IFIP TC2 Conference on Simulation Programming Languages. May 1967.
R. Dedekind. *What are and what should the numbers be?* Friedr. Vieweg & Sohn, 1888. Translated by David E. Joyce, Clark University, Dec. 2005; https://mathcs.clarku.edu/~djoyce/numbers/dedekind.pdf
S. Feferman. *Is the Continuum Hypothesis a definite mathematical problem?* Exploring the Frontiers of Independence. Harvard lecture series. 2011.
E. Fermi. *Artificial radioactivity produced by neutron bombardment* Nobel Lecture. December 12, 1938.

J. Girard. *Interprétation fonctionnelle et Élimination des coupure de l'arithmétique d'ordre supérieur* 1972.
K. Gödel. *The completeness of the axioms of the functional calculus of logic* Monatshefte für Mathematik und Physik 3. 1930
K. Gödel. *On formally undecidable propositions of Principia Mathematica* Monatshefte für Mathematik und Physik. 1931. Translation in *From Frege to Gödel: A Source Book in Mathematical Logic*. Harvard University Press. 1967.
M. Gordon. *Isabelle and HOL* Cambridge Automated Reasoning. 2016.
A. E. Haas. *Uber die elektrodynamische Bedeutung des Planckschen Strahlungsgesetzes und uber eine neue Bestimmung des elektrischen Elementarquantums und der dimension des wasserstoffatoms. Sitzungsberichte der kaiserlichen* Akademie der Wissenschaften in Wien. 1910.
C. Hewitt. *Planner: A Language for Proving Theorems in Robots* IJCAI. 1969.
C. Hewitt, P. Bishop, and R. Steiger. *A Universal Modular Actor Formalism for Artificial Intelligence* IJCAI. 1973.
C. Hewitt and R. Atkinson. *Specification and Proof Techniques for Serializers* IEEE Journal on Software Engineering. January 1979.
C. Hewitt. *Strong Types for Direct Logic.* HAL Archive; 2017-2019. https://hal.archives-ouvertes.fr/hal-01566393
C. Hewitt. *Citadels provide performance and security for citizens and companies alike: Verifiably ending use of sensitive citizen information for mass surveillance can foster (international) commerce and law enforcement* Social Science Research Network. Working Paper 2836282. 2016-2019. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2836282
C. Hewitt. *Building and Deploying Scalable Intelligent Systems by 2025* Video of Stanford University EE380 Colloquium on January 23, 2019. http://web.stanford.edu/ class/ee380/Abstracts/190123.html
C. Hewitt and J. Woods assisted by Jane Spurr. *Inconsistency Robustness, 2nd Edition* Studies in Logic. 2019.
A. Hermann. *Arthur Erich Haas* The Columbia Encyclopedia, 6th ed. 2008.
T. Hoare *Monitors: An Operating System Structuring Concep*t CACM. October 1974.
ISO. *Programming languages -- C++* ISO/IEC 14882:2017. December 2017.
G. Kreisel. *Informal Rigour and Completeness Proofs* Problems in the Philosophy of Mathematics. Elsevier. 1967.
T. Kuhn. *The Structure of Scientific Revolutions. 50th anniversary edition* University of Chicago Press. 2012.
I. Lakatos. *Proofs and Refutations.* Cambridge University Press. 1976.
M. Löb. *Solution of a problem of Leon Henkin* Journal of Symbolic Logic. Vol. 20. 1955.
P. Maddy. *What do we want a foundation to do? Comparing set-theoretic, category-theoretic, and univalent approaches* Reflections on Foundations: Univalent Foundations, Set Theory and General Thoughts. 2018.
A. Maltzev. *Untersuchungen aus dem Gebiete der mathematischen Logik,*Matematicheskii sbornik 1, *1936.* English translation in *The metamathematics of algebraic systems: collected papers,1936-1967* North-Holland. 1971.
P. Martin-Löf. *An intuitionistic theory of types in Twenty-Five Years of Constructive Type Theory* Oxford University Press. 1998.
J. McCarthy, P. Abrahams, D. Edwards, T. Hart, and M. Levin, *LISP 1.5 Programmer's Manual* 1962.
R. Milner. *Elements of interaction: Turing award lecture* CACM. January 1993.
G. H. Moore. *The Emergence of First-Order Logic* History and Philosophy of Modern Mathematics. Minnesota Studies in the Philosophy of Science. Volume XI. 1988.
L. Paulson. *A machine-assisted proof of Gödel's incompleteness theorems for the theory of hereditarily finite sets*. Review of Symbolic Logic 7 3. 2014.
G. Plotkin. *A powerdomain construction* SIAM Journal of Computing. September 1976.
G. Priest. *Yablo's Paradox* Analysis 57. 1997.
B. Russell. *Les paradoxes de la logique* Revue de métaphysique et de morale. 1906.
B. Russell. *Mathematical Logic as Based on the Theory of Types* American Journal of Mathematics. 30 (3). 1908.
A. Turing. *On Computable Numbers, with an Application to the Entscheidungsproblem* Proceedings of the London Mathematical Society. 2. 42. 1936.
A. N, Whitehead and B. Russell. *Principia Mathematica,* Cambridge University Press 1910-1913.
L. Wittgenstein. *Remarks on the Foundations of Mathematics, Revised Edition* Basil Blackwell. 1978.
J. Woods. *Errors of Reasoning. Naturalizing the Logic of Inference* Studies in Logic. 2013.
J. Woods. *How paradox fares in Inconsistency Robust Logic and beyond: Computational and naturalized approaches* Inconsistency Robustness, 2nd Edition. Studies in Logic. 2019.
S. Yablo. *Truth and reflection* Journal of Philosophical Logic. 14 (2). 1985.