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 categorical 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 and computational systems which are not strongly-typed necessarily provide opportunities for cyberattack.

Cyberattackers have severely damaged national, corporate, and individual security as well causing hundreds of billions of dollars of economic damage. [Sobers 2019] 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].

Kurt 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 strongly-typed universal model of digital computation (including implementation of Scalable Intelligent Systems) up to a unique isomorphism. Strongly-typed Actors provide the foundation for tremendous improvements in cyberdefense.

Index Terms—uniquely categorical theories, strong types, Scalable Intelligent Systems, Actor Model of Computation, Alonzo Church, Haskell Curry, Richard Dedekind, Kurt Gödel, Thomas Kuhn, Martin Löb, Gordon Plotkin, Bertrand Russell, Alan Turing, Ludwig Wittgenstein, John Woods, Stephen Yablo

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, set of elements of a type, computable procedures, and Actors, as well as the theories of these structures.

† 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/
In a strongly typed mathematical theory, every proposition, mathematical term, and program expression has a type. 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 \textit{The Liar} which could be constructed as a paradoxical fixed point using the mapping \( p \mapsto \neg p \) (notation from [Bourbaki 1939-2016]), except for the requirement that each proposition must have an order beginning with 1st-order. Since \( p \) is a propositional variable in the mapping, \( \neg p \) has order one greater than the order of \( p \). \textbf{Thus because of orders on propositions, there is no paradoxical fixed point for the mapping \( p \mapsto \neg p \) which if it existed could be called I'mFalse such that I'mFalse \( \iff \neg \text{I'mFalse} \).} Unfortunately in addition to attaching orders to propositions, Russell 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 \textit{I'mUnprovable} [Gödel 1931] into Russell’s foundations for mathematics infers a contradiction as follows:

```
“Let us suppose [Gödel 1931 was correct and therefore] I prove the unprovability (in Russell’s system) of [Gödel’s I'mUnprovable] \( P \); \text{[i.e., \( \vdash _{\text{Russell}} \neg \forall_{\text{Russell}} P \) where \( P \iff \forall_{\text{Russell}} P \)]} then by this proof I have proved \( P \) \text{[i.e., \( \vdash _{\text{Russell}} P \) because \( P \iff \forall_{\text{Russell}} P \)]}. Now if this proof were one in Russell’s system \text{[i.e., \( \vdash _{\text{Russell}} \forall_{\text{Russell}} P \)]} — I should in this case have proved at once that it belonged \text{[i.e., \( \vdash _{\text{Russell}} P \)]} and did not belong \text{[i.e., \( \forall_{\text{Russell}} \neg P \) because \( \neg P \iff \forall_{\text{Russell}} P \)]} to Russell’s system. But there is a contradiction here! \text{[i.e., \( \vdash _{\text{Russell}} P \) and \( \forall_{\text{Russell}} \neg P \)]} ...

This is what comes of making up such propositions.” [emphasis added] [Wittgenstein 1978]
```

\textit{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 instead of the theory \( \text{Russell} \), which is an extension of Russell’s theory by adding the natural numbers induction axiom as stated in [Gödel 1931]. In this way, \textbf{Gödel dodged the point of Wittgenstein’s criticism.}

Technically, the result in [Gödel 1931] was as follows: Consistent[\( \text{Russell} \)] \( \Rightarrow \vdash _{\text{Russell}} \forall_{\text{Russell}} P \) where \( P \iff \forall_{\text{Russell}} P \) and Consistent[\( \text{Russell} \)] if an only if there is no proposition \( \Psi \) such that \( \vdash _{\text{Russell}} \forall \neg \Psi \). However, Wittgenstein was understandably taking it as a given that \( \text{Russell} \) is consistent because it formalized standard mathematical practice and had been designed to block known paradoxes (such as \textit{The Liar}) using orders on propositions. Consequently, Wittgenstein elided the result in [Gödel 1931 to \( \vdash _{\text{Russell}} \forall_{\text{Russell}} P \). His point was that \( \text{Russell} \) is consistent provided that the proposition \( \vdash _{\text{Russell}} \forall_{\text{Russell}} P \) is not added to \( \text{Russell} \). Wittgenstein was justified because the standard theory of natural numbers is arguably consistent because it has a model. [Dedekind 1888] See [Shanker 1988] for further discussion of Wittgenstein on Gödel’s results.

According to [Russell 1950]: “A new set of puzzles has resulted from the work of Gödel, especially his article [Gödel 1931], in which he proved that in any formal system [with recursively enumerable theorems] it is possible to construct sentences of which the truth [i.e., provability] or falsehood [i.e., unprovability] cannot be decided within the system. Here again we are faced with the essential necessity of a hierarchy [of sentences], extending upwards ad infinitum, and logically incapable of completion.” [Urquhart 2016] Construction of Gödel’s \textit{I'mUnprovable} is blocked because the mapping
\(\Psi \mapsto \not \models \Psi\) does not have a fixed point because the order of \(\not \models \Psi\) is one greater than the order of \(\Psi\) since \(\Psi\) is a propositional variable.

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 [Gödel 1930] (meaning that every countable inconsistent set of propositions has a finite inconsistent subset). Compactness is false of the standard theory of natural numbers for the following reason: if \(k\) is a natural number then the set of propositions of the form \(i > k\) where \(i\) 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 \(\infty\)) for a natural number, which is a “monster” [Lakatos 1976] because \(\infty\) is larger than every standard natural number.
  - Every 1st-order theory \(\mathcal{T}\) that can formalize its own provability has a model \(\mathcal{M}\) with a Gödelian “monster” element proposition \(\Gamma\) that proves \(\mathcal{T}\) inconsistent (i.e. \(\models_{\mathcal{M}} \neg \Gamma \land \Gamma\)) by the following proof:

    According to [Gödel 1931], \(\not \models_{\mathcal{T}} \text{Consistent}(\mathcal{T})\) and consequently because of the 1st-order model “completeness” theorem [Gödel 1930] there must be some model \(\mathcal{M}\) of \(\mathcal{T}\) in which \(\text{Consistent}(\mathcal{T})\) 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 theory with recursively enumerable theorems 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 allow 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 \(x: \tau\) means that \(x\) is of type \(\tau\). For example, \(0:N\) expresses that \(0\) is of type \(N\), which is the type of a natural number. Types are intensional, i.e., if \(x:t_1 \Leftrightarrow x:t_2\) for every \(x\) does not mean that \(t_1 = t_2\) where \(t_1\) and \(t_2\) are types. Burali-Forti/Girard paradox is blocked because for every type \(\tau\), \(\neg \tau: \tau\) and is \(\tau\) is of type \(\text{TypeOf}<\tau>\).

- \(t_2^{t_1}\) 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 **nondeterministic computable** procedures from $t_1$ into $t_2$ where $t_1$ and $t_2$ are types whereas $t_1 \rightarrow t_2$ is the deterministic procedures. For example, $[ \rightarrow ]$ is the type all partial nondeterministic procedures of no argument into the type of $\text{Boolean}$. If $p: [ \rightarrow ] \rightarrow \text{Boolean}$, then $p \bullet []$ starts a computation by providing input $[]$ to procedure $p$ which might return $\text{True}$ or return $\text{False}$. *It also might happen that $p \bullet []$ does not return a value.*

• $[t_1, t_2]$ is type of pairs of $t_1$ and $t_2$ where $t_1$ and $t_2$ are types. For example, $[N, \text{Boolean}]$ is the type of pairs whose first is a natural number and whose second is a Boolean.

• $\text{PropositionOfOrder} < i >$ is type of a proposition of order $i$ where $i: N_+$ and $N_+$ is the type of positive natural numbers. For example, $\text{PropositionOfOrder} < 1 >$ is the type of propositions of order 1.
  
  o $\text{Proposition} \; \Psi$ means $\exists [i: N_+] \; \Psi: \text{PropositionOfOrder} < i >$
  
  o $P \; \text{predicateOn} \; t$ means $\exists [i: N_+] \; P: \text{PropositionOfOrder} < i >^t$

• $t_2^P$ is the type of $t$ restricted to $P$ where $t$ is a type and $P$ is a predicate. For example, replacement for types is expressed using restriction, i.e., the range of a function $f: t_2^t$ is $t_2 \exists y \rightarrow \exists [x: t_1] \; y = f[x]$.

• $\text{TypeOf} < t >$ is the type of the type $t$. For example, $N: \text{TypeOf} < N >$.

**Types are constructed bottom-up from types that are categorically axiomatized up to a unique isomorphism. Type checking is linear in the size of the proposition, 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 procedures. *It is followed later by a section on how to precisely define Actors, which are a fundamental generalization of procedures.*

**Theory of Classical Computable Procedures**

The theory of classical **nondeterministic computable** procedures (e.g. Lambda Expression [Church 1931] and Turing Machine [Turing 1936]), will be denoted by the name $\rightarrow$. (This article also discusses a more general theory of computation called $\text{Actor}$.) $\text{Eval} < t >: [\text{Expression} < t > \; \text{in} \; \text{Environment}] \rightarrow t$ is a procedure [McCarthy et. al. 1962] that corresponds to a universal Turing machine [Turing 1936] as follows:

  o $\text{Eval} < \text{Expression} < t > \;; \; \text{x} > = \text{Eval} < t > \;; \; \text{x \; in \; EmptyEnvironment}$
  
  o $\text{Eval} < \text{Identifier} < t > \;; \; \text{x \; in \; e \; Environment} > = \text{Lookup} [\text{x \; in \; e}]$
  
  o $\text{Eval} < \text{Application} < t \_ \_ \_; \; \text{t} \_ \_ \_; \; \text{e} \_ \_ \_; \; \text{x} \_ \_ \_; \; \text{y} \_ \_ \_; \; \text{z} \_ \_ \_; \; \text{operator} \_ \_ \_; \; \text{operand} \_ \_ \_; \; \text{in} \; \text{e} \; \text{Environment} >$

  \hspace{1cm} (\text{Eval} < \text{Expression} < t \_ \_ \_; \; \text{t} \_ \_ \_; \; \text{operator} \_ \_ \_; \; \text{in} \; \text{e} >) \cdot (\text{Eval} < \text{Expression} < t \_ \_ \_; \; \text{operator} \_ \_ \_; \; \text{operand} \_ \_ \_; \; \text{in} \; \text{e} >))$

  \hspace{1cm} // apply the value of operator to the value of operand

  o $\text{Eval} < \text{Procedure} \_ \_ \_; \; \text{t} \_ \_ \_; \; \text{t} \_ \_ \_; \; \text{e} \_ \_ \_; \; \text{x} \_ \_ \_; \; \text{y} \_ \_ \_; \; \text{z} \_ \_ \_; \; \text{body} \_ \_ \_; \; \text{in} \; \text{e} \; \text{Environment} >$

  \hspace{1cm} \text{x} \_ \_ \_; \; \text{t} \_ \_ \_; \; \text{eval} \_ \_ \_; \; \text{body} \_ \_ \_; \; \text{in} \; \text{Bind} \_ \_ \_; \; \text{x} \_ \_ \_; \; \text{y} \_ \_ \_; \; \text{z} \_ \_ \_; \; \text{in} \; \text{e} >$

  \hspace{1cm} // eval body in a new environment with x1 bound to x2 as an extension of e

In order to implement recursion, the lambda calculus has the primitive $\text{Fix}$ such that

$\forall [ F : \text{Functional} < t_1, t_2 > ] \; \text{Fix} < t_1, t_2 > [ F ] = F \bullet \text{Fix} < t_1, t_2 > [ F ]$ where

**Functional** $< t_1, t_2 > \equiv (t \rightarrow t) \rightarrow (t \rightarrow t)$

The theory $\rightarrow$ has the following induction axiom:

$\forall [ \text{PredicateOn} \_ \_ \_; \; t_1 \_ \_ \_; \; F : \text{Functional} < t_1, t_2 > ]$

$((P \bullet F [ \bot < t_1, t_2 > ] \land \forall [ g : t_1 \_ \_ \_; \; \text{P} [ g ] \Rightarrow P \bullet [ F \bullet \text{Fix} < t_1, t_2 > [ g ] ] ] \Rightarrow P \bullet [ F \bullet \text{Fix} < t_1, t_2 > [ F ] ] ) \Rightarrow \text{P} [ \text{Fix} < t_1, t_2 > [ F ] ]$

where $\bot < t_1, t_2 > = x : t_1 \rightarrow \bot < t_1, t_2 > \cdot x$
Metatheory of the theory $\rightarrow$

$\mathsf{Meta} \leftrightarrow \mathsf{Imp}$ is a meta theory of $\rightarrow$ for proving theorems about $\rightarrow$, which directly expresses provability of a proposition $\Psi$ in using $\vdash \rightarrow \Psi$. (Gödel numbers cannot be used to represent propositions because there are not enough Gödel numbers to represent all uncountably many propositions that are instances of the induction axiom.)

Proof Checkers in the theory $\rightarrow$

A proof checker $\text{pc}: \text{ProofChecker} \leftarrow \rightarrow$ [cf. Gordon, Milner and Wadsworth 1979] 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 $\rightarrow$) means that $\text{pc}$ is proof checker such that proposition $\Psi_1$ infers proposition $\Psi_2$ in the theory $\rightarrow$ (written $\Psi_1 \vdash_{\text{pc}} \Psi_2$) such that:

$$\forall \text{[Proposition } \Psi_1, \Psi_2 \text{]} \ (\Psi_1 \vdash_{\text{pc}} \Psi_2) \iff \exists \text{[pc: ProofChecker} \leftarrow \rightarrow \text{]} \ (\Psi_1 \vdash_{\text{pc}} \Psi_2)$$

Proof checking in the theory $\rightarrow$ is computationally decidable because:

$$\forall \text{[Proposition } \Psi_1, \Psi_2 \text{]} \ (\Psi_1 \vdash_{\text{pc}} \Psi_2) \iff \text{pc}[\Psi_1, \Psi_2] = \text{True}$$

where $\text{pc}[\Psi_1, \Psi_2]$ means the invocation of procedure $\text{pc}$ with arguments $\Psi_1$ and $\Psi_2$. For example, a proof checker for the induction axiom is as follows:

InductionChecker $\text{[}\Psi, \Psi_2\text{]} \equiv \forall \text{[i]} \ (P[\text{[}\Psi\text{]}][\text{[i]}] \wedge \forall \text{[i]} \ P[\text{[i]}] \Rightarrow P[\text{[i]} + 1])$ then $P_2 = \forall \text{[i]} \ P[\text{[i]}]$, else False

Note that InductionChecker correctly checks uncountably many instances of each of the theory $\rightarrow$ induction axioms.

There are uncountable proof checkers in the theory $\rightarrow$ 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 $\leftarrow \rightarrow$ where $t$ is a type and $c:t$ such that

ForAllEliminationChecker $\leftarrow \rightarrow$ $[c] \leftarrow \rightarrow$ $\equiv \forall \text{[i]} \ (\forall \text{[x:t]} \ P[\text{[x]}])$ then $P_2=P[c]$, else False

Consequently, $\forall \text{[x:t]} \ P[\text{[x]}] \vdash \rightarrow$ ForAllEliminationChecker $\leftarrow \rightarrow$ $[c] \rightarrow P[c]$

Types and propositions of the theory $\rightarrow$

Types and propositions of the theory $\rightarrow$ are axiomatized in terms of each other.

The following axioms hold for $\text{TypeIn} \leftarrow \rightarrow$ (the type of types in the theory $\rightarrow$) because types are intensional:

- $\forall \text{[x:N]} \ \text{PropositionOfOrder} \leftarrow \rightarrow$ $\equiv \forall \text{[x]} (\forall \text{[i]}:N \rightarrow \text{TypeIn} \leftarrow \rightarrow)
- \forall \text{[t_1,t_2,t_3,t_4:TypeIn} \leftarrow \rightarrow \text{]} \ (t_1,t_2) \iff t_1=t_2 \wedge t_3=t_4$
- $\forall \text{[t_1,t_2,t_3,t_4:TypeIn} \leftarrow \rightarrow \text{]} \ t_1 \rightarrow t_2 \iff t_3 \rightarrow t_4 \iff t_1=t_2 \wedge t_3=t_4$
- $\forall \text{[t_1,t_2:TypeIn} \leftarrow \rightarrow \text{]} ; P_1 \text{predicateOn} \rightarrow t_1, P_2 \text{predicateOn} \rightarrow t_2 \iff t_1 \rightarrow P_1 \iff t_2 \rightarrow P_2 \iff t_1=t_2 \wedge P_1=P_2$

For example, $(N\rightarrow N): \text{TypeIn} \leftarrow \rightarrow$, etc.
The following induction axiom holds, which has uncountable instances:
\[
\forall [P \text{ predicateOn} \rightarrow \text{TypeIn} \langle \rightarrow \rangle] \\
(P[M] \land \forall [i:N] \exists [P[\text{OrderProof} \rightarrow \langle i \rangle]]) \\
\quad \land \forall [t_1, t_2: \text{TypeIn} \langle \rightarrow \rangle] P[t_1] \land P[t_2] \Rightarrow P[t_1 \rightarrow t_2] \\
\quad \land \forall [t: \text{TypeIn} \langle \rightarrow \rangle] P[t] \Rightarrow P[t \rightarrow Q] \\
\quad \Rightarrow \exists [t': \text{TypeIn} \langle \rightarrow \rangle] P[t']
\]

**Theorem.** Unique categoricity of \(\text{TypeIn} \langle \rightarrow \rangle\), i.e., if \(M\) is a type satisfying the theory \(\rightarrow\), then there is a unique isomorphism \(I\) between \(\text{TypeIn} \langle \rightarrow \rangle\) and \(\text{TypeIn}_M\langle \rightarrow \rangle\) defined as follows:

- \(I[t_1, t_2] = I[t_1, I[t_2]]_M\)
- \(I[t_2 \rightarrow t_2] = I[t_2] \rightarrow I[t_2]\)
- \(I[t \equiv P] = I[t]_M \equiv [P]\)

The following induction axiom holds for propositions of the theory \(\rightarrow\), which has uncountable instances:
\[
\forall [i:N, P \text{ predicateOn} \rightarrow \text{Proposition} \langle \rightarrow \rangle] \\
\quad \land \forall [t: \text{TypeIn} \langle \rightarrow \rangle; x_1, x_2: t] P[x_1=x_2] \land \forall [t_1, t_2: \text{TypeIn} \langle \rightarrow \rangle; x: t_2] P[x:t] \\
\quad \land \forall [\text{Proposition} \rightarrow \Psi] P[\Psi] \Rightarrow P[-\Psi] \land \forall [\text{Proposition} \rightarrow \Psi, \Psi_1, \Psi_2] P[\Psi_1] \land P[\Psi_2] \Rightarrow P[\Psi_1 \land \Psi_2] \\
\quad \land \forall [t: \text{TypeIn} \langle \rightarrow \rangle; Q \text{ predicateOn} \rightarrow t] (\forall [x:t] P[Q[x]]) \Rightarrow P[\forall [x:t] Q[x]] \\
\quad \Rightarrow \forall [\text{Proposition} \rightarrow \Psi] P[\Psi]
\]

**Theorem.** Propositions of the theory \(\rightarrow\) are characterized up to a unique isomorphism.

**Inference in the theory \(\rightarrow\)**

Inferential soundness means that a theorem in \(\rightarrow\) can be used in proofs in \(\rightarrow\).

**Theorem.** Inferential Soundness of the theory \(\rightarrow\), i.e.,
\[
\vdash_{\text{Meta} \langle \rightarrow \rangle} \forall [\text{Proposition} \rightarrow \Psi] (\vdash_{\rightarrow} \Psi) \Rightarrow \Psi
\]

**Proof.** If \(\vdash_{\rightarrow} \Psi\), then \(\Psi\) holds in Model\(\langle \rightarrow \rangle\).

A consequence of Inferential Soundness is that unrestricted cut-elimination does not hold for the theory \(\rightarrow\).

**Theorem.** Deduction for the theory \(\rightarrow\), i.e., the following holds:
\[
\vdash_{\text{Meta} \langle \rightarrow \rangle} \forall [\text{Proposition} \rightarrow \Phi, \Psi] (\vdash_{\rightarrow} \Phi) \Rightarrow (\Phi \vdash_{\rightarrow} \Psi)
\]

**Proof.** Suppose \(\vdash_{\rightarrow} \Phi \Rightarrow \Psi\) and consequently \(\Phi \Rightarrow \Psi\) by Inferential Soundness. Further suppose \(\Phi\) is provably total procedure \(\text{pc1: ProofChecker} \langle \rightarrow \rangle\) and consequently \(\Phi \vdash_{\rightarrow} \Psi\) by InferenceIntroduction.

On the other hand suppose \(\Phi \vdash_{\rightarrow} \Psi\). Further suppose \(\Phi\). Then \(\Psi\) by ChainingForImplication and consequently \(\Phi \vdash_{\rightarrow} \Psi\) by ImplicationIntroduction.

**Theorem** Inferential Adequacy, i.e., \(\vdash_{\text{Meta} \langle \rightarrow \rangle} \forall [\text{Proposition} \rightarrow \Psi] (\vdash_{\rightarrow} \Psi) \Rightarrow \vdash_{\rightarrow} \vdash_{\rightarrow} \Psi\)

**Proof.** Suppose \(\vdash_{\rightarrow} \Psi\). Let \(\vdash_{\rightarrow} \Psi\) so that \(\text{pc1} \vdash \Psi\) = \text{True}. Then a provably total procedure \(\text{pc2: ProofChecker} \langle \rightarrow \rangle\) can be defined such that \(\text{pc2} \vdash \Psi\) = \text{True}. Consequently, \(\vdash_{\rightarrow} \vdash_{\rightarrow} \Psi\).

**Theorem:** If \(M\) is a type satisfying the axioms of the theory \(\rightarrow\), then there is a unique isomorphism \(M_{\text{Model} \langle \rightarrow \rangle}\).
The theory → is computationally and inferentially undecidable

The predicate Halt can be defined as follows on deterministic expressions:
\[ \text{Halt}[x]\text{:Deterministic} \triangleq \exists [y: \text{N}] y = \text{Eval}[x] \]

**Definitions.**
- **Decider**: \( t \triangleq \text{Total} \text{Deterministic} \text{[t]} \rightarrow \text{Boolean} \)
- **t: String**: \( t \text{x} \rightarrow \exists [s: \text{String}: \text{Deterministic} \text{[t]} \rightarrow \text{Boolean}] x: [s] \) where \([s]\) is the abstraction of \( s \)
- **BExpression**: \( \text{Deterministic} \text{[Boolean]} \rightarrow \text{String} \)
- **BProcedure**: \( (\text{BExpression} \rightarrow \text{Boolean}) \rightarrow \text{String} \)

**Theorem.** Halt is computationally undecidable [Church 1935, Turing 1936], i.e.,
\[ \exists [d: \text{Decider} \text{[BProcedure]}] \forall [x: \text{BExpression}] \text{d}[x] = \text{True} \iff \text{Halt}[x] \]

**Proof.** Suppose to obtain a contradiction that
\( \exists [d: \text{Decider} \text{[BProcedure]}] \forall [x: \text{BExpression}] \text{d}[x] = \text{True} \iff \text{Halt}[x] \).

Define **Opp** as follows where ( and ) are used to delimit expressions.
\[ \text{Opp}[p: \text{BProcedure}, \text{BExpression}] \rightarrow \text{Boolean}, x: \text{Boolean} \rightarrow \text{true} \]

\[ \text{d}([p([p, x]])) ?? \text{True then LoopForever}[\text{[ }], \text{False then True"}] \]

Consider **Opp** for \( \text{Halt}[\text{True}] \) to obtain a contradiction. By hypothesis for \( d \), there are two cases:

1. \( \text{d}([\text{Opp}[\text{True}])] = \text{True} \). Thus, \( \neg \text{Halt}([\text{Opp}[\text{True}]]) \) by the definition of **Opp**, which contradicts the hypothesis for \( d \).
2. \( \text{d}([\text{Opp}[\text{True}])] = \text{False} \). Thus, \( \text{Halt}([\text{Opp}[\text{True}]]) \) by the definition of **Opp**, which contradicts the hypothesis for \( d \).

Thus both cases are contradictory and \( d \) does not exist.

**Theorem.** Whether a proposition is a theorem of → is computationally undecidable [Church 1935, Turing 1936], i.e., there does not exist a decider \( d \) for propositions of the theory → such that
\[ \forall [\text{Proposition} \rightarrow \Psi] \text{d}([\Psi]) = \text{True} \iff \vdash \neg \Psi \]

**Proof.** Follows immediately from the computational undecidability of the halting problem because of the following: \( \forall [x: \text{Deterministic} \rightarrow \text{Boolean}] \text{Halt}[x] \iff \vdash \neg \text{Halt}[x] \)

**Theorem.** The theory → is inferentially undecidable, e.g.,
\[ \exists [x: \text{Deterministic} \rightarrow \text{Boolean}] (\vdash \text{Halt}[x]) \land \vdash \neg \text{Halt}[x] \]

**Proof.** Suppose to obtain a contradiction that the theory → is inferentially decidable and consequently
\[ \forall [x: \text{Deterministic} \rightarrow \text{Boolean}] (\vdash \neg \text{Halt}[x]) \lor (\vdash \neg \neg \text{Halt}[x]) \]

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

**Theorem.** There is a proposition of the theory → that is true of the natural numbers but unprovable in →, e.g., \( \exists [x: \text{Deterministic} \rightarrow \text{N}] \neg \text{Halt}[x] \land \vdash \neg \text{Halt}[x] \)

**Proof.** By inferential undecidability let \( x \) be such that \( (\vdash \neg \text{Halt}[x]) \land \vdash \neg \neg \text{Halt}[x] \). Therefore \( \neg \text{Halt}[x] \) because \( \text{Halt}[x] \equiv \vdash \neg \text{Halt}[x] \).

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

The theory → 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 → violates this assumption because there are uncountable instances of the induction axiom. Uncountability of raises the
following question: What axioms of the theory → can be expressed in text, i.e., in the theory →↑String; i.e.,
the theory → abstracted from strings.

The theory →↑String has the following induction axiom, which has countable instances because strings
are countable: ∀[P predicateOn→↑String N] (P[0] ∧ ∀[j:N] P[j]⇒P[j+1]) ⇒ ∀[j:N] P[j]

Definitions.
- Total→↑String≡(N→1t)∀[y:t] f[y]=y
- ProvedTotal→↑String≡((N→1t)↑String)∀[y:t] ⊢↑String f:Total→↑String
- Onto→↑String≡(N→1t)∀[y:t] ∃[x:N] f[x]=y
- ProvedEnumerate→↑String≡ProvedTotal→↑String f:Onto→↑String

Theorem. Theorem ↔↑String is computationally enumerable, i.e., there is a procedure

Theorems of type ProvedEnumerate→↑String<Theorem ↔↑String

Corollary. ProvedTotal→↑String is computationally enumerable, i.e., there is a procedure

Proof. Define the procedure Diagonal as follows:

Diagonal[i:N]≡1+(ProvedTotals[i]1[i]

Lemma. Diagonal:ProvedTotal→↑String

Proof. Suppose i:N. Let

f:ProvedTotal→↑String=ProvedTotals[i] and let j:N=f[i]. Therefore Diagonal[i]=1+j.

Consequently, ⊢↑String Diagonal:Total→↑N.

Lemma. ¬Diagonal:ProvedTotal→↑String

Proof. Diagonal differs from every ProvedTotal→↑String enumerated by ProvedTotals.

Theorem. The theory →↑String is inconsistent [Church 1934], i.e.,

∃[Proposition→↑String Ψ] ⊢↑String Ψ∧¬Ψ

Proof. Let Ψ=Diagonal:ProvedTotal→↑String

The upshot is that the theory → is algorithmically inexhaustible, i.e., nonalgorithmic creativity will be
forever required to develop new → axioms abstracted from strings thereby reinforcing the intuition
behind [Franzén, 2004]. According to [Church 1934], inconsistency of the theory →↑String 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.

V. ACTOR MODEL

[Church 1932] and [Turing 1936] developed equivalent models of computation 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. For example, λ expressions [Church 1932] play an important role in programming languages.
Also, Turing Machines [Turing 1936] inspired development of the stored program sequential computer and Logic Programs [Hewitt 1969] are fundamental to Scalable Intelligent Systems.

*Computation that cannot be done by \( \lambda \) Calculus, Nondeterministic Turing Machines, or pure Logic Programs*

As shown below, Actor machines can perform computations that a no \( \lambda \) expression, nondeterministic Turing Machine or pure Logic Program can implement.

There is an *always-halting* Actor machine that can compute an integer of unbounded size. This is accomplished using an Actor with a variable \( \text{count} \) that is initially 0 and a variable \( \text{continue} \) initially True. The computation is begun by concurrently sending two messages to the Actor machine: a stop request that will return an integer and a go message that will return \( \text{Void} \). The Actor machine operates as follows:

- When a stop message is received, return \( \text{count} \) and set \( \text{continue} \) to False for the next message received.
- When a go message is received:
  - If \( \text{continue} \) is True, increment \( \text{count} \) by 1, send this Actor machine a go message in a hole of the region of mutual exclusion, and afterward return \( \text{Void} \).
  - If \( \text{continue} \) is False, return \( \text{Void} \).

*Theorem*. There is no \( \lambda \) 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.”

*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. Then the set of propositions

\[ \exists i : N \text{--Output}[i] \] is inconsistent but every finite subset \( S \) is consistent because the Actor machine output might be larger than any output in \( S \).

Interactive computation has 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), indeterminacy in Actor systems illustrate 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, "interface", 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:

```
```

A manager for a readers-writer scheduler has the following interface:

```
ReadersWriterManager interface getScheduler → ReadersWriter,
      upgrade[ReadersWriterManager] → Void
```

**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 an implementation `ReadPriority` that implements the interface `ReadersWriterManager` with a facet that implements `ReadersWriter` (cf. [Amborn 2004, Brinch Hansen 1996, Crahen 2002, Hoare 1974]).
ReadPriority [[aDatabase:ReadersWriter]] implements ReadersWriterManager ↦

Local(FIFO( writersQ, readersQ ),
  // writersQ, readersQ are queues of suspended activities
  Crowd( reading ), // reading is a crowd of readers
  AtMostOne( writing )); // writing is a crowd with at most 1

Invariant(Nonempty[ writing ] ⇒ isEmpty[ reading ])
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: AllEmpty[ 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 implementation, e.g. ReadPriority, differs from a class [Dahl and Nygaard 1967] as follows:
4. An implementation can use multiple other implementations (thereby avoiding having to copy and paste code) using qualified names to prevent ambiguity [cf. ISO 2017].
5. An implementation cannot be subclassed [Dahl and Nygaard 1967] in order to prevent impersonation by other types.
6. An invariant for an Actor must hold when it is created and when entering/leaving a continuous section of a region of mutual exclusion.
7. Strong types are the foundation of Actor communication. For example, if x is of type ReadPriority, then x.getScheduler means ReadPriority.send[x, getScheduler]

Types manage crypto without requiring programming by application programmers.

Theorem. Readers exclude writers. Suppose manager1 is ReadPriority[database1]. After manager1 has sent a write request to database1, it will not send another request to until it has received a response because the invariant Nonempty[ writing ] ⇒ isEmpty[ reading ] holds as follows:
• 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 has been processed.
Theorem. \texttt{ReadPriority[database]} forwards messages to database. Starvation of activities suspended in readers and writers as is prevented in a \texttt{ReadPriority} implementation as follows:

- An activity in readers progresses when
  1. A read to the database is started by another activity
  2. If writers 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 writers progresses when
  1. If readers 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 readers when another activity starts a read to the database.

Axiomatization of Actors up to a unique isomorphism

Let \texttt{x[e]} be the behavior of Actor \texttt{x} at local event \texttt{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.

The theory \texttt{Actor} categorically axiomatises Actors using the following axioms where \(\sim\) (read as “precedes”) is transitive and irreflexive and Info[\texttt{x}] is the information in the Actor addresses of \texttt{x}:

- Primitive Actors
  - \(\forall [i : \mathbb{N}] i : \texttt{Actor}\) // natural numbers are Actors
  - \(\forall [x_1, x_2 : \texttt{Actor}] [x_1, x_2] : \texttt{Actor}\) // a 2-tuple of Actors is an Actor
- Event ordering
  - \(\forall [c : \texttt{Com}] \exists [x] [x : \texttt{Actor}, c : \texttt{Com}] \exists x_1. \text{sent}[cx_1] \) // every communication was sent by an Actor
  - \(\forall [c : \texttt{Com}] \forall [x_1, x_2] \text{Received}[x_1] \land \text{Received}[x_2] \Rightarrow x_1 = x_2\)
    // a communication is received at most once
  - \(\forall [x] \text{Initial}[x] \Rightarrow \text{Received}[x] \sim \text{After}[x]\)
  - \(\forall [x, x_1, x_2] \text{Received}[x_1] \lor \text{Received}[x_2] \Rightarrow \text{Received}[x_1] \not\sim \text{Received}[x_2]\)
  - \(\forall [c, x_1, x_2] \exists [e] \text{Event}[e] \not\sim e_1 \not\sim e_2\) // There are only finitely many events
    // in \(\sim\) between two events.
- An Actor’s behavior change
  - \(\forall [x, c] \text{Received}[x_1] \Rightarrow \text{Received}[x_2] \Rightarrow x \text{received}[c] = x \text{initial}\)
  - \(\forall [x, c] \text{Received}[x_1] \Rightarrow \text{Received}[x_2] \Rightarrow x \text{event}[c] = x \text{after}[c]\)
  - \(\forall [x] \text{New}[c] \exists s \exists x \text{sent}[c]\)
  - \(\forall [x] \text{New}[c] \not\exists x \text{created}[c] \subseteq \text{Info}[c] \cup \text{Info}[\text{Received}[c] \cup \text{Info}[\text{New}[c]]\)
    // New[c] is addresses of Actors created processing c
  - \(\forall [x, x_1] \text{New}[c] \Rightarrow x_1 \in \text{New}[c] \Rightarrow \bot = \text{Info}[x_1] \cap (\text{Info}[c] \cup \text{Info}[\text{Received}[c] \cup \text{Info}[\text{New}[c] - x_1])\)
    // info about the address of a newly created Actor does not provide any
    // information about addresses of other current Actors
  - \(\forall [x, x_1] \text{New}[c] \land e \sim \text{Received}[c_2] \Rightarrow \bot = \text{Info}[x_1] \cap \text{Info}[e]\)
    // info about the address of a newly created Actor does not provide any
    // information about addresses in previous events.
ο ∀[x₁,x₂:Actor, c₁,c₂:Comv] c₁≠c₂∧x₁∈New[c₁]∧x₂∈New[c₂] ⇒ ⊥=Info[x₁]∩Info[x₂]

// info about the address of a newly created Actor does not provide any information
// about address of any other newly created Actor
ο ∀[x:Actor, c:Comv] Let processing = (Info[c]∪Info[Receivedₜ[c₂]] )∪Info[x=created[c]]

// processing is information about addresses that is in c, available in the Actor
// when c was received and in Actors created while processing c
  in (Info[x=after[c]]∈processing ∧ Info[x=sent[c]]∈processing)

• Actor Induction
∀[x:Actor, P: predicateOn, A:Actor], Behavior
(P[ᵲ[x:=initial]] ∧ ∀[c:Comv] P[ᵲ[x=received[c]]]  
  ⇒ P[ᵲ[x=after[c]]] ) ⇒ ∀[c:Comv] P[ᵲ[x=received[c]]] ∧ P[ᵲ[x=after[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 provide a response which may be a thrown exception.

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

Foundational 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. Each theory T has the following properties:
• T is uniquely categorical for Model<Ω>, i.e., if X satisfies the axioms of T, then is X isomorphic to Model<Ω>, by a unique isomorphism.
• T is sound, i.e., (⊥⇒Φ) ⇒ Φ
• For all propositions Ψ of T and p:ProofChecker<Ω>, ⊢ₜΨ is computationally decidable.
• T is not compact, i.e., it is not the case that for every inconsistent set of T propositions S that S has a finite inconsistent subset.

Mathematical Foundations for Computer Science
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
• 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 of elements of a type,
groups, lambda calculus, and Actors up to a unique isomorphism side-by-side. For example, the ordinals \( O \) can be axiomatized using strong types so that there is just one model up to a unique isomorphism, which is more general than 1st-order set theory because Boolean\(^O\) is not part of the cumulative hierarchy of sets.

- **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, set of elements of a type, 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, set of elements of a type, groups, lambda calculus, and Actors.

- **Guidance** is for practitioners 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, set of elements of a type, 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 the theories \( \rightarrow \) and \( \text{Actor} \) is based on the way that they are inductively constructed bottom-up.

- **Monsters** [Lakatos 1976] are unwanted elements in models of classical mathematical theories. \( \text{Actor} \) precisely characterizes what is digitally computable leaving no room for “monsters” in models. Having a model up to a unique isomorphism in classical mathematical theories is crucial for cybersecurity.

- **Inferential completeness** is the ability to directly express all inference of classical mathematics. The ordinals \( O \) can be uniquely categorically axiomatized in the theory \( O \) (using induction for the ordinals in a way analogous to induction on \( N \) in the theory \( \mathbb{N} \)) that can directly express proofs of theorems of classical mathematics including [Wiles 1995]. As shown, in this article, additional axioms for Actors are needed to axiomatize digital computation.

Intuitive categorical inductive axiomatizations of natural numbers, propositions, types, ordinals, set of elements of a type, 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 [Sobers 2019, Perlroth, Sanger and Shane 2019] cries out for a paradigm shift.
Nature of Paradigm Shifts

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.”

Shifting Away from 1st-order Logic Foundations

Computer Science must shift from 1st-order logic 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
- being able to infer each and every proposition (including nonsense) from an inconsistency in an empirical theory even though it may not be apparent 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.”

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.

Shifting Away from Models of Computation That Are Not Strongly-typed

Influenced by Turing Machines [Turing 1936], current computer systems are not strongly-typed leaving them open to cyberattacks [Hewitt 2019]. Strongly-typed Actors can directly model and
implement all digital computation. Consequently, strongly-typed architecture can be extended to microprocessors providing strongly-typed computation all the way to hardware.

The Establishment has made numerous mistakes during paradigm shifts.

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 “Asonium” and “Hesperium”, which were actually mixtures of barium, krypton and other elements. [Fermi 1938]

How the Computer Science cybersecurity crisis will proceed is indeterminate

Possibilities going forward include the following:

- continue to 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

Much recent work has centered on constructive type theory (e.g. [Coquand 1986]) which has type $t_1 \rightarrow t_2$, which is the type of deterministic computable procedures on $t_1$ into $t_2$, but does not have $t_2 \rightarrow t_1$, which is the type of all functions on $t_1$ into $t_2$. Also, constructive type theory relies on the premise that $\Psi$ is a proposition of theory $T$ if and only if $\Psi$ is a theorem of $T$ with the unfortunate consequence that type checking is computationally undecidable and it is difficult to reason about unprovable propositions.

HOL Light [Harrison 2017] allows more general types than constructive type theory although it is not strongly typed and does not have explicit parameterized types, e.g., a proposition does not have an order, which raises issues with taking fixed points. Also, HOL Light considers two propositions to be equal if they are logically equivalent with the unfortunate consequence that it is difficult to reason about propositions that happen to be logically equivalent. For example, all theorems are considered to be equal and can consequently be freely substituted for each other in all terms and propositions.

VIII. CONCLUSION

This article strengthens the position of Computer Science cybersecurity 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]. Automaton 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, and Gordon Plotkin and were extremely helpful in developing ideas in this article. Richard Waldinger made very helpful comments. Natarajan 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. John Perry provided extensive comments throughout the article. John Woods suggested an improved title.

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:

- \( \neg \Psi_1, \Psi_1 \land \Psi_2: \text{PropositionOrder}^{\langle \text{ii} \rangle} \) where \( \Psi_1, \Psi_2: \text{PropositionOrder}^{\langle \text{ii} \rangle} \) and \( i: \mathbb{N}_+ \)
- \( (x_1=x_2): \text{PropositionOrder}^{\langle \text{i1} \rangle} \) where \( x_1, x_2: \text{Term}^{\langle t \rangle} \) and \( t \) is a type
- \( (x: t): \text{PropositionOrder}^{\langle \text{i1} \rangle} \) where \( t \) is a type
- \( P^n x]: \text{PropositionOrder}^{\langle i+1 \rangle} \) where \( x: \text{Term}^{\langle t \rangle}, t \) is a type and \( P: \text{Term}^{\langle \text{Proposition}^{\langle \text{i} \rangle} \rangle} \) and \( i: \mathbb{N}_+ \)
- \( (\Psi_1 \land \Psi_2): \text{PropositionOrder}^{\langle \text{ii} \rangle} \) where \( i: \mathbb{N}_+ \) and \( \Psi_1, \Psi_2: \text{PropositionOrder}^{\langle \text{ii} \rangle} \)
- \( (\Psi_1 \vdash p, \Psi_2): \text{PropositionOrder}^{\langle \text{ii} \rangle} \) where \( p: \text{Term}^{\langle \text{ProofChecker} \rangle}, T: \text{Theory}, \Psi_1, \Psi_2: \text{PropositionOrder}^{\langle \text{ii} \rangle} \) and \( i: \mathbb{N}_+ \)
- \( [s]: \text{PropositionOrder}^{\langle \text{ii} \rangle} \) is abstraction of \( s \) where \( s: \text{String}^{\langle \text{PropositionOrder}^{\langle \text{ii} \rangle} \rangle} \) with no free variables and \( i: \mathbb{N}_+ \)
- \( [\Psi]: \text{String}^{\langle \text{PropositionOrder}^{\langle \text{ii} \rangle} \rangle} \) is quotation of \( \Psi \) where \( \Psi: \text{PropositionOrder}^{\langle \text{ii} \rangle} \) and \( i: \mathbb{N}_+ \).
Procedural *Expression* is a discrimination of the following:
- o *x:* Expression $\langle t \rangle$ where *x:* Constant$\langle t \rangle$ and $t$ is a type
- o *x:* Expression $\langle t \rangle$ where *x:* Identifier$\langle t \rangle$ and $t$ is a type
- o $[e_1, e_2]:$ Expression $\langle [t_2, t_2] \rangle$ where $e_1:$ Expression $\langle t_1 \rangle$, $e_2:$ Expression $\langle t_2 \rangle$, and $t_1$ and $t_2$ are types
- o $(e_1 \, ?? \, True \, then \, e_2, \, False \, then \, e_3):$ Expression $\langle t \rangle$ where $e_1:$ Expression $\langle Boolean \rangle$, $e_2,e_3:$ Expression $\langle t \rangle$ and $t$ is a type
- o $(\lambda [x : t_1] \, y):$ Expression $\langle t_1 \to t_2 \rangle$ where *x:* Identifier$\langle t_1 \rangle$, *y:* Expression$\langle t_2 \rangle$ and $t_1$ and $t_2$ are types
- o $x,m:$ Expression $\langle t_2 \rangle$ where $m:$ Expression $\langle t_1 \rangle$, *x* is an Actor with a message handler with signature of type Expression $\langle t_1 \to t_2 \rangle$, and $t_1$ and $t_2$ are types
- o $I[x_1, \ldots, x_n]:$ Expression $\langle I \rangle$ where $I$ is an Actor implementation and $x_1, \ldots, x_n$ are expressions.
- o $[s]:$ Expression $\langle t \rangle$ is abstraction of $s$ where $s:$ String$\langle \text{Expression} \langle t \rangle \rangle$ with no free variables and $t$ is a type
- o $[x]:$ String$\langle \text{Expression} \langle t \rangle \rangle$ is quotation of $x$ where *x:* Expression $\langle t \rangle \mid \text{String}$, $i:N_+$ and $t$ is a type.

Mathematical *Term* is a discrimination of the following patterns:
- o *x:* Term $\langle t \rangle$ where *x:* Constant$\langle t \rangle$ and $t$ is a type
- o *x:* Term $\langle t \rangle$ where *x:* Variable$\langle t \rangle$ and $t$ is a type
- o $[x_1, x_2]:$ Term $\langle [t_1, t_2] \rangle$ where *x_1:* Term $\langle t_2 \rangle$, *x_2:* Term $\langle t_2 \rangle$, and $t_1$ and $t_2$ are types
- o $(x_1 \, ?? \, True \, then \, x_2, \, False \, then \, x_3):$ Term $\langle t \rangle$ where $x_1:$ Term $\langle Boolean \rangle$, $x_2,x_3:$ Term $\langle t \rangle$ and $t$ is a type
- o $([x : t_1] \mapsto y):$ Term $\langle t_2 \, t_1 \rangle$ where *x:* Variable$\langle t_1 \rangle$, *y:* Term$\langle t_2 \rangle$ and $t_1$ and $t_2$ are types
- o $[f]:$ Term $\langle t_2 \rangle$ where $f:$ Term $\langle t_2 \, t_1 \rangle$, *x:* Term$\langle t_1 \rangle$, and $t_1$ and $t_2$ are types
- o $[s]:$ Term $\langle t \rangle$ is abstraction of $s$ where $s:$ String$\langle \text{Term} \langle t \rangle \rangle$ with no free variables and $t$ is a type
- o $[x]:$ String$\langle \text{Term} \langle t \rangle \rangle$ is quotation of *x* where *x:* Term $\langle t \rangle \mid \text{String}$, $i:N_+$ 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.

**Burali-Forti/Girard** [Burali-Forti 1897, Girard 1972, Coquand 1986]
Although each ordinal $\alpha$ can be strictly embedded as a well-founded order in the ordinals $\mathcal{O}$ and $\alpha = {}_{\mathcal{O}} \mathcal{O} \mathcal{B} \mathcal{P} \mapsto \alpha$ as Ordinals, $\neg \mathcal{O}(\mathcal{O} \mathcal{B} \mathcal{P} \mapsto \beta : \mathcal{O})$ because $\neg \mathcal{O} : \mathcal{O}$, which blocks the paradox. Also, there is no universal type in strongly-typed theories, which blocks [Girard 1972] for [Martin-Löf 1971].

**Russell** [Russell 1902]
- o 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.
- o Russell’s paradox for predicates is resolved as follows: The mapping $P \mapsto \neg P[\mathbb{P}]$ has no fixed point because $\neg P[\mathbb{P}]$ has order one greater than the order of $P$ because $P$ is a predicate variable.
Wittgenstein [Wittgenstein 1978]
Wittgenstein’s Paradox is blocked because the mapping \( \Psi \mapsto \nu \Psi \) does not have a fixed point (contra [Gödel 1931]) because the order of \( \nu \Psi \) is greater than the order of \( \Psi \) since \( \Psi \) is a propositional variable.

Curry [Curry 1941]
Curry’s Paradox is blocked because the mapping \( p \mapsto (p \Rightarrow \Psi) \) does not have a fixed point because the order of \( p \Rightarrow \Psi \) is greater than the order of \( p \) since \( p \) is a propositional variable.

Löb [Löb 1955]
Löb’s Paradox is blocked because the mapping \( p \mapsto ((\vdash p) \Rightarrow \Psi) \) does not have a fixed point because the order of \( (\vdash p) \Rightarrow \Psi \) is greater than the order of \( p \) since \( p \) is a propositional variable.

Yablo [Yablo 1985]
Yablo’s Paradox is blocked because the mapping \( P \mapsto (\forall [i, j > i : N] \neg P[j]) \) does not have a fixed point because the order of \( \forall [i, j > i : N] \neg P[j] \) is one 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\( i \bullet \)\( k \) meaning that the string \( s \) characterizes the integer \( k \) as follows where \( i : N \):  

- \( \text{Berry} \equiv (\Term \text{PropositionOfOrder} \downarrow i \downarrow N) \text{\textbf{\downarrow String}} \)
- \( \text{Characterize} \equiv [s : \text{Berry}, k : N] \equiv \forall [x : N] [s][x] \Leftrightarrow x = k \)

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:

\[ \text{BerryString} : \text{Berry} \equiv (\forall [i, j > i : N] \neg \text{Characterize} [s, j]) \]

Note that
- \( \text{Length}[\text{BerryString}] < 100. \)
- \( \text{Berry} : \exists s \Rightarrow \text{Length}[s] < 100 \) is finite.
- Therefore, \( \text{BerryNumber} \) is finite where \( \text{BerryNumber} \equiv \exists i : N \exists j \exists [s : \text{Berry}] \text{Length}[s] < 100 \land \text{Characterize}[i][s, j] \)
- \( \exists [i : N] i : \text{BerryNumber} \) because is \( N \) is infinite.
- \( \text{LeastBerry} \equiv \exists [\text{BerryNumber}] \)
- \( \exists [\text{BerryString}, \text{LeastBerry}] = \forall [s : \text{Berry}] \text{Length}[s] < 100 \Rightarrow \neg \text{Characterize}[s, \text{LeastBerry}] \)

However \( \text{BerryString} : \text{Berry} \equiv (\forall [i, j > i : N] \neg \text{Characterize} [s, \text{LeastBerry}]) \) cannot be substituted for \( s : \text{Berry} \). Consequently, the Berry Paradox as follows does not hold:

[\text{BerryString}, \text{LeastBerry}] \Leftrightarrow \neg \text{Characterize}[\downarrow \text{BerryString}, \text{LeastBerry}]
**APPENDIX: ORDINALS AND NATURAL NUMBERS**

**Theory of Natural Numbers**

The mathematical theory \( \mathbb{N} \) that axiomatises the Natural Numbers \( \mathbb{N} \) has the following axioms building on [Dedekind 1888]:

- \( 0 : \mathbb{N} \)  // 0 is of type \( \mathbb{N} \)
- \( +_1 : \mathbb{N}^\mathbb{N} \)  // +_1 (successor) is of type \( \mathbb{N}^\mathbb{N} \)
- \( i [i : \mathbb{N}] +_1 [i] = 0 \)  // 0 is not a successor
- \( \exists [i, j : \mathbb{N}] +_1 [i] = +_1 [j] \Rightarrow i = j \)  // +_1 is 1 to 1

In addition, the theory \( \mathbb{N} \) has the following induction axiom, which has uncountable instances:

\[
\forall [P \text{ predicate on } \mathbb{N}] (P[0] \land \forall [i : \mathbb{N}] P[i] \Rightarrow P[+_1[i]]) \Rightarrow \forall [j : \mathbb{N}] P[j]
\]

**Theorem** [cf. Dedekind 1888]: If \( \mathcal{M} \) be a type satisfying the axioms of the theory \( \mathbb{N} \), then there is a unique isomorphism \( I : \mathcal{M}^{\text{Model } \mathbb{N}} \) defined as follows:

- Define by induction on \( \text{TypeIn}^{\mathcal{M}^{\text{Model } \mathbb{N}}} \)
  - \( I[0] = 0 \)
  - \( I[+_1[j]] = +_1^\mathcal{M}[I[j]] \)
  - if \( x : [t_1, t_2] \), then \( I[x] = [I[1^{\text{st}}[x]], I[2^{\text{nd}}[x]]] \)
  - if \( x : [t_2, t_2] \), then \( I[x] = y : I[1^{\text{st}}[y]] \)  // inductive hypothesis for \( I \) on \( t_2 \)

\( I \) is a unique isomorphism because of the following:

- \( I \) is defined on \( \text{TypeIn}^{\mathcal{M}^{\text{Model } \mathbb{N}}} \)
- \( I \) is 1-1
- \( I \) is onto \( \mathcal{M} \)
- \( I \) is a homomorphism
- \( I^\mathcal{M} \) is a homomorphism
- If \( g \) is an isomorphism of \( \text{Model}^{\mathcal{M}^{\text{Model } \mathbb{N}}} \) with \( \mathcal{M} \), then \( g = I \)

**Corollary** There are no infinite numbers in models of the theory \( \mathbb{N} \), e.g., if \( \mathcal{M} \) satisfies the axioms of the theory \( \mathbb{N} \) for \( \mathbb{N} \), then \( \exists [j : \mathcal{M}] \forall [i : \mathbb{N}] i < j \)

**Theory of Ordinals**

The theory \( \mathcal{O} \) that axiomatises the ordinals \( \mathcal{O} \) has the following axioms in addition to the axioms for the theory \( \mathbb{N} \):

- \( 0 : \mathcal{O} \)  // 0 is an ordinal
- \( \exists [\alpha : \mathcal{O}] \alpha : 0 \)  // 0 has no predecessor
- \( +_1 : \mathcal{O} \times \mathcal{O} \rightarrow \mathcal{O} \)  // \( +_1[\alpha] = \alpha + 1 \)
- \( \forall [\alpha, \beta : \mathcal{O}] \alpha : \beta \Leftrightarrow \alpha : \beta \)
- \( \forall [\alpha, \beta, \gamma : \mathcal{O}] \alpha : \beta \land \beta : \gamma \Rightarrow \alpha : \gamma \)
- \( \forall [\alpha, \beta : \mathcal{O}] \exists [\gamma : \mathcal{O}] \alpha : \beta \land \beta : \gamma \Rightarrow \alpha : \gamma \)
- \( \forall [\alpha : \mathcal{O}] \exists [\beta : \mathcal{O}] \alpha : \beta \land \beta : +_1[\alpha] \)
- \( \forall [\alpha : \mathcal{O}] f : \text{Nondecreasing}^{\mathcal{O}} ) U_{\alpha f} : \mathcal{O} \)  // \( U_{\alpha f} \) is limit of range of \( f \) on \( \alpha \)
• ∀[α: O, f: Nondecreasing <O>] \bigcup_0 f = 0
• ∀[α: O, f: Nondecreasing <O>] \bigcup_{α+1} f = f[α] + 1
• ∀[α, β: O; f: Nondecreasing <O>] f[β] ≤ \bigcup_α f // \bigcup_α f includes all of range of f on α
• ∀[α: O; f: Nondecreasing <O>] (∀y < α) f[y] ≤ β) ⇒ (\bigcup_α f) ≤ β
  // \bigcup_α f is minimum of range of f on α

• ∀[α: O] \omega_α: O
• \omega_0 = N
• ∀[α: O] 1to1[\omega_{α+1}, \text{Boolean}^\omega]\alpha] // jump in cardinality
• ∀[α, γ: O] 1to1[γ, \text{Boolean}^\omega]\alpha] ⇒ \omega_{α+1} ≤ γ // minimal jump in cardinality
• ∀[α: O] (\exists[β: O] α = +1[β]) ⇒ \omega_α = \bigcup_β (β → ω_β)

In addition, the theory O has the following induction axiom, which has uncountable instances:
∀[P \text{ predicateOn}_O O] (P[0] ∧ ∀[α: O] ∀[β: β < α: O] P[β] ⇒ P[α])) ⇒ ∀[α: O] P[α]

Theorem: O is well-ordered by <, i.e. \exists[f: O^N] ∀[i: N] f(i+1) < f[i]

Theorem: If M is a type satisfying the axioms of the theory O, then there is a unique isomorphism
I: M^{Model <O>} defined as follows:
• Define by induction on TypeIn <O>
  o I[O] = M
  o I[t_1, t_2] = I[t_1], I[t_2] \_M
  o I[t^t_1] = I[t^t_2]
  o ∀[P \text{ predicateOn}_O TypeIn <O>, t: TypeIn <O>] I[t \exists P] = I[t \exists I[P]
• Define by induction on TypeIn <O>
  o Define by induction on O
    • I[0] = 0_M
    • I[+1(α)] = +1_M[I[α]]
    • I[ω_α] = \omega_α^M
    • otherwise I[\bigcup_α f] = \bigcup_α^M[I[f]
  o if x: [t_1, t_2], then I[x] = [I[1\_[x]], I[2\_[x]]] \_M
  o if x: t^t_2, then I[x] = y: I[t_2] → I[x[I^{-1}[y]]] // inductive hypothesis for I on t_2

REFERENCES

C. Burali-Forti. (1897) A question on transfinite numbers A Source Book in Mathematical Logic Harvard University Press.

C. Burali-Forti. *Una questione sui numeri transfiniti* Rendiconti del Circolo Matematico di Palermo. 1897


K. Gödel. *The completeness of the axioms of the functional calculus of logic* Monatshefte für Mathematik und Physik 3, 1930


C. Hewitt. *Strong Types for Direct Logic.* HAL Archive; 2017-2019. [https://hal.archives-ouvertes.fr/hal-01566393](https://hal.archives-ouvertes.fr/hal-01566393)


