For Cybersecurity, Computer Science Must Rely on Strongly-Typed Actors

Carl Hewitt†

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 *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. **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* $\Leftrightarrow \neg$*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 *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; [*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]

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 $\mathbb{Russell}$, which is an extension of Russell's theory by adding the natural numbers induction axiom as stated in [Gödel 1931]. In this way, **Gödel dodged the point of Wittgenstein's criticism.**

Technically, the result in [Gödel 1931] was as follows: Consistent[$\mathbb{Russell}$]$\Rightarrow \vdash_{\mathbb{Russell}} \nvdash_{\mathbb{Russell}} P$ where $P \Leftrightarrow \nvdash_{\mathbb{Russell}} P$ and Consistent[$\mathbb{Russell}$] 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 $\mathbb{Russell}$ 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 $\mathbb{Russell}$ is consistent provided that the proposition $\vdash_{\mathbb{Russell}} \nvdash_{\mathbb{Russell}} P$ is **not** added to $\mathbb{Russell}$. 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 *I'mUnprovable* is blocked because the mapping

$\Psi \mapsto \nvdash \Psi$ does **not** have a fixed point because the order of $\nvdash \Psi$ is one greater than the order of Ψ since Ψ 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 ∞) 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 $\mathcal{M}$ with a Gödelian "monster" element proposition Γ that proves $\mathbb{T}$ inconsistent (i.e. $\vDash_{\mathcal{M}} \vdash_{\mathbb{T}} \Gamma \wedge \neg \Gamma$) by the following proof: 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 $\mathcal{M}$ 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 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: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 *intensional*, i.e., if x:t_1⇔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 t, $\neg t{:}t$ and is t is of type $TypeOf\triangleleft t\triangleright$.
- $t_2^{t_1}$ 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_1 where t_1 and t_2 are types whereas $t_1 \rightarrow_1 t_2$ is the deterministic procedures. For example, *[]→Boolean* is the type all partial nondeterministic procedures of no argument into the type of *Boolean*. If p:*[]→Boolean*, then p▪[ ] starts a computation by providing input [ ] to procedure p which might return True or return False. *It also might happen that* p▪[ ] *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, Boolean]* is the type of pairs whose first is a natural number and whose second is a Boolean.
- *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.
  - Proposition Ψ means ∃[i:N_+] Ψ:*PropositionOfOrder*◁i▷
  - P predicateOn t means ∃[i:N_+] P:*PropositionOfOrder*◁i▷^{t}
- t∋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:$t_2^{t_1}$ is t_2∋y↦∃[x:t_1] y=f[x].
- *TypeOf*◁t▷ is the type of the type t. For example, *N: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 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 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 →. (This article also discusses a more general theory of computation called Actor.) Eval◁t▷:[*Expression*◁t▷ in *Environment*]→t is a procedure [McCarthy et. al. 1962] that corresponds to a universal Turing machine [Turing 1936] as follows:

- Eval◁*Expression*◁t▷▷▪[x] ≡ Eval◁t▷▪[x in EmptyEnvironment]
- Eval◁*Identifier*◁t▷▷▪[x in e:*Environment*] ≡ Lookup[x in e]
- Eval◁*Application*◁t_1, t_2▷▷▪[(operator▪operand) in e:*Environmment*] ≡
  (Eval◁*Expression*◁t_1→t_2▷▷▪[operator in e]]▪(Eval◁*Expression*◁t_1▷▷▪[operand in e])
  // apply the value of operator to the value of operand
- Eval◁*Procedure*◁t_1, t_2▷▷▪[(x1↦body) in e:*Environmment*] ≡
  x2:t_1 ↦ Eval◁*Expression*◁t_2▷▷▪[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 order to implement recursion, 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)

The theory → has the following induction axiom:

∀[P predicateOn→ t_1→t_2, F:*Functional*◁t_1,t_2▷]
(P⟦F▪[⊥◁t_1,t_2▷]⟧ ∧ ∀[g:t_1→t_2] P⟦g⟧ ⇨ P⟦F▪[Fix◁t_1, t_2▷▪g]⟧) ⇨ P⟦Fix◁t_1, t_2▷▪F⟧
where ⊥◁t_1,t_2▷ ≡ x:t_1 ↦ ⊥◁t_1,t_2▷▪x

Metatheory of the theory →

𝕄𝕖𝕥𝕒◁→▷ is a meta theory of → for proving theorems about →, 𝕨𝕙𝕚𝕔𝕙 ***directly*** **expresses provability of a proposition Ψ in using** ⊢→Ψ. (*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 →

A proof checker pc:*ProofChecker*◁→▷ [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 →) means that pc is proof checker such that proposition Ψ_1 infers proposition Ψ_2 in the theory → (written $\Psi_1 \vdash\frac{\text{pc}}{\rightarrow} \Psi_2$) such that:

$$\forall[\underline{\text{Proposition}}\ \Psi_1,\Psi_2]\ (\Psi_1 \vdash_{\rightarrow} \Psi_2) \Leftrightarrow \exists[\text{pc}:\mathit{ProofChecker}\lhd\rightarrow\rhd]\ \ \Psi_1 \vdash\frac{\text{pc}}{\rightarrow}\Psi_2$$

Proof checking in the theory → is computationally decidable because:

$$\forall[\underline{\text{Proposition}}\ \Psi_1,\Psi_2],\ \text{pc}:\mathit{ProofChecker}\lhd\rightarrow\rhd](\Psi_1 \vdash\frac{\text{pc}}{\rightarrow} \Psi_2) \Leftrightarrow \text{pc}_\blacksquare[\Psi_1,\Psi_2]=\underline{\text{True}}$$

where pc■[Ψ_1,Ψ_2] means the invocation of procedure pc with arguments Ψ_1 and Ψ_2. For example, a proof checker for the induction axiom is as follows:

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

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

There are uncountable proof checkers in the theory → 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 **??** (∀[x:*t*] P[x]) <u>then</u> Ψ_2=P[c], <u>else</u> <u>False</u>

Consequently, $(\forall[x:t]\ P[x]) \vdash\frac{\text{ForAllEliminationChecker}\lhd t\rhd[c]}{\rightarrow} P[c]$

Types and propositions of the theory →

Types and propositions of the theory → are axiomatized in terms of each other.

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

- *N*:*TypeIn*◁→▷ // *N* is type of natural numbers
- ∀[i:N_+] *PropositionOfOrder*◁i▷:*TypeIn*◁→▷
- $\forall[t_1,t_2,t_3,t_4:\mathit{TypeIn}\lhd\rightarrow\rhd]\ [t_1,t_2]=[t_3,t_4] \Rightarrow t_1=t_2 \wedge t_3=t_4$
- $\forall[t_1,t_2,t_3,t_4:\mathit{TypeIn}\lhd\rightarrow\rhd]\ t_1\rightarrow t_2=t_3\rightarrow t_4 \Rightarrow t_1=t_2 \wedge t_3=t_4$
- ∀[t_1,t_2:*TypeIn*◁→▷; P_1 <u>predicateOn</u>→t_1, P_2 <u>predicateOn</u>→t_2] t_1∋P_1=t_2∋P_2 ⇨ $t_1=t_2$ ∧ P1=P_2

For example, (*N*→*N*):*TypeIn*◁→▷, etc.

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

$\forall$[P $\underline{\text{predicateOn}}_{\rightarrow}$ TypeIn◁→▷]

($P⟦N⟧ \wedge \forall[i{:}N_+]$ P[$\mathit{PropositionOfOrder}_{\rightarrow}$◁i▷]]

$\wedge\ \forall[t_1,t_2{:}\mathit{TypeIn}$◁→▷] $P⟦t_1⟧ \wedge P⟦t_1⟧ \Rightarrow P⟦[t_1,t_2]⟧$

$\wedge\ \forall[t_1,t_2{:}\mathit{TypeIn}$◁→▷] $P⟦t_1⟧ \wedge P⟦t_2⟧ \Rightarrow P⟦t_1 \rightarrow t_2⟧$

$\wedge\ \forall[t{:}\mathit{TypeIn}$◁→▷, Q $\underline{\text{predicateOn}}_{\rightarrow}$ t] $P⟦t⟧ \Rightarrow P⟦t\underline{\ni}Q⟧$)

$\Rightarrow \forall[t{:}\mathit{TypeIn}$◁→▷] $P⟦t⟧$

Theorem Unique categoricity of TypeIn◁→▷, i.e., if $\mathcal{M}$ is a type satisfying the theory →, then there is a unique isomorphism I between TypeIn◁→▷ and $\mathit{TypeIn}_{\mathcal{M}}$◁→▷ defined as follows:

- $I[[t_1,t_2]] \equiv [I[t_1], I[t_2]]_{\mathcal{M}}$
- $I[t_1 \rightarrow t_2] \equiv I[t_1] \rightarrow I[t_2]$
- $I[t\underline{\ni}P] \equiv I[t] \ni_{\mathcal{M}} I[P]$

The following induction axiom holds for propositions of the theory →, *which has uncountable instances*:

($\forall$[i:N_+, P $\underline{\text{predicateOn}}_{\rightarrow}$ $\mathit{PropositionOfOrder}_{\rightarrow}$◁i▷]

$\forall[t{:}\mathit{TypeIn}$◁→▷; $x_1,x_2{:}t$] $P⟦x1{=}x2⟧ \wedge \forall[t_1,t_2{:}\mathit{TypeIn}$◁→▷; $x{:}t_2$] $P⟦x{:}t⟧$

$\wedge\ \forall$[$\underline{\text{Proposition}}_{\rightarrow}$ Ψ] $P⟦\Psi⟧ \Rightarrow P⟦\neg\Psi⟧ \wedge \forall$[$\underline{\text{Proposition}}_{\rightarrow}$$\Psi_1,\Psi_2$] $P⟦\Psi_1⟧ \wedge P⟦\Psi_2⟧ \Rightarrow P⟦\Psi_1 \wedge \Psi_2⟧$

$\wedge\ \forall[t{:}\mathit{TypeIn}$◁→▷; Q $\underline{\text{predicateOn}}_{\rightarrow}$ t] $(\forall[x{:}t]\ P⟦Q⟦x⟧⟧) \Rightarrow P[\forall[x{:}t]\ Q⟦x⟧]$)

$\Rightarrow \forall$[$\underline{\text{Proposition}}_{\rightarrow}$Ψ] $P⟦\Psi⟧$

Theorem. Propositions of the theory → are characterized up to a unique isomorphism.

Inference in the theory →

Inferential soundness means that a theorem in → can be used in proofs in →.

Theorem: Inferential Soundness of the theory →, i.e.,

$$\vdash_{\mathrm{Meta}◁\rightarrow▷} \forall[\underline{\text{Proposition}}_{\rightarrow}\Psi]\ (\vdash_{\rightarrow}\Psi) \Rightarrow \Psi$$

Proof. If $\vdash_{\rightarrow}\Psi$, then Ψ holds in Model◁→▷.

A consequence of Inferential Soundness is that unrestricted cut-elimination does not hold for the theory →.

Theorem: Deduction for the theory →, i.e., the following holds:

$$\vdash_{\mathrm{Meta}◁\rightarrow▷} \forall[\underline{\text{Proposition}}_{\rightarrow}\Phi,\Psi]\ (\vdash_{\rightarrow}\Phi\Rightarrow\Psi) \Leftrightarrow (\Phi\vdash_{\rightarrow}\Psi)$$

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

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

Theorem Inferential Adequacy, i.e., $\vdash_{\mathrm{Meta}◁\rightarrow▷}\forall[\underline{\text{Proposition}}_{\rightarrow}\Psi]\ (\vdash_{\rightarrow}\Psi) \Rightarrow \vdash_{\rightarrow}\vdash_{\rightarrow}\Psi$

Proof: Suppose $\vdash_{\rightarrow}\Psi$. Let $\vdash\frac{\mathrm{pc1}}{\rightarrow}\Psi$ so that pc1▪[Ψ]=$\underline{\text{True}}$**.** Then a provably total procedure pc2:$\mathit{ProofChecker}$◁→▷ can be defined such that pc2▪[$\vdash_{\rightarrow}\Psi$]=$\underline{\text{True}}$. Consequently, $\vdash_{\rightarrow}\vdash_{\rightarrow}\Psi$.

Theorem : If $\mathcal{M}$ is a type satisfying the axioms of the theory →, then there is a unique isomorphism $\mathcal{M}^{\mathit{Model}◁\rightarrow▷}$.

The theory $\rightarrow$ *is computationally and inferentially undecidable*

The predicate Halt can be defined as follows on deterministic expressions:

Halt[x:*Deterministic*◁$\mathbb{N}$▷] ≡ ∃[y:$\mathbb{N}$] y=Eval▪[x]

Definitions.

- *Decider*◁*t*▷ ≡ *Total*◁*Deterministic* ◁ *[t]*→*Boolean*▷▷
- *t*↾*String* ≡ *t*∋x↦∃[s:*String*◁*t* ▷] x:⌊s⌋ where⌊s⌋is the abstraction of s
- *BExpression* ≡ *Deterministic*◁*Boolean* ▷↾*String*
- *BProcedure* ≡ (*BExpression* $\rightarrow_1$ *Boolean*)↾*String*

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

∄[d:*Decider*◁*BProcedure* ▷] ∀[x:*BExpression*] d▪[x]=True ⇔ Halt[x]

Proof. Suppose to obtain a contradiction that

d:*Decider*◁*BProcedure* ▷ and ∀[x:*BExpression*▷] d▪[x]=True ⇔ Halt[x].

Define Opp as follows *where* ⦅ *and* ⦆ *are used to delimit expressions.*

Opp ≡ [p:*[BProcedure*, *BExpression]* → *Boolean*, x:*Boolean*] ↦

d▪[⦅p▪[p, x]⦆] **??** True then LoopForever▪[ ], False then True"]

Consider Opp▪[Opp, True] to obtain a contradiction. By hypothesis for d, there are two cases:

1. d▪[⦅Opp▪[Opp, True⦆]=True. Thus, ¬Halt[⦅Opp▪[Opp, True]⦆] by the definition of Opp, which contradicts the hypothesis for d.
2. d▪[⦅Opp▪[Opp, True⦆]=False. Thus, Halt[⦅Opp▪[Opp, 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 $\rightarrow$ is computationally undecidable [Church 1935, Turing 1936], i.e., there does not exist a decider d for propositions of the theory $\rightarrow$ such that

∀[Proposition$_\rightarrow$ Ψ] d▪[Ψ]=True ⇔ $\vdash_\rightarrow$Ψ

Proof. Follows immediately from the computational undecidability of the halting problem because of the following: ∀[x:*Deterministic*◁*Boolean*▷] Halt[x]⇔ $\vdash_\rightarrow$Halt[x]

Theorem. The theory $\rightarrow$ is inferentially undecidable, e.g.,

∃[x:*Deterministic*◁*Boolean*▷] ($\nvdash_\rightarrow$ Halt[x]) ∧ $\nvdash_\rightarrow$¬Halt[x]

Proof. Suppose to obtain a contradiction that the theory $\rightarrow$ is inferentially decidable and consequently

∀[x:*Deterministic*◁*Boolean*▷] ($\vdash_\rightarrow$ Halt[x]) ∨ ($\vdash_\rightarrow$¬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 $\rightarrow$ that is true of the natural numbers but unprovable in $\rightarrow$, e.g., ∃[x:*Deterministic*◁$\mathbb{N}$▷] ¬Halt[x] ∧ $\nvdash_\rightarrow$¬Halt[x]

Proof. By inferential undecidability let x be such that ($\nvdash_\rightarrow$Halt[x])∧$\nvdash_\rightarrow$¬Halt[x]. Therefore ¬Halt[x] because Halt[x]⇨ $\vdash_\rightarrow$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 $\rightarrow$ *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 $\rightarrow$ violates this assumption because there are uncountable instances of the induction axiom. Uncountability of raises the following question: What axioms of the theory $\rightarrow$ can be expressed in text, i.e., in the theory $\rightarrow\upharpoonleft\mathit{String}$, i.e., the theory $\rightarrow$ abstracted from strings.

The theory $\rightarrow\upharpoonleft\mathit{String}$ has the following induction axiom, **which has countable instances** because strings are countable**:** $\forall[P\ \underline{\text{predicateOn}}_{\rightarrow\upharpoonleft_{String}}\mathbb{N}]\ (P[\![0]\!] \wedge \forall[j{:}\mathbb{N}]\ P[\![j]\!] \Rightarrow P[\![+_1[j]]\!]) \Rightarrow \forall[j{:}\mathbb{N}]\ P[\![j]\!]$

Definitions.

- $\mathit{Total}\triangleleft t\triangleright \equiv (\mathbb{N}\rightarrow_1 t)\underline{\exists}f\mapsto\forall[x{:}\mathbb{N}]\ \exists[y{:}t]\ f_{\blacksquare}[x]=y$
- $\mathit{ProvedTotal}_{\rightarrow\upharpoonleft_{String}} \equiv\ ((\mathbb{N}\rightarrow_1\mathbb{N})\upharpoonleft\mathit{String})\underline{\exists}f\mapsto\vdash_{\rightarrow\upharpoonleft_{String}} f{:}\mathit{Total}\triangleleft t\triangleright$
- $\mathit{Onto}\triangleleft t\triangleright \equiv (\mathbb{N}\rightarrow_1 t)\underline{\exists}f\mapsto\forall[y{:}t]\ \exists[x{:}\mathbb{N}]\ f_{\blacksquare}[x]=y$
- $\mathit{ProvedEnumerator}_{\rightarrow\upharpoonleft_{String}}\triangleleft t\triangleright \equiv \mathit{ProvedTotal}_{\rightarrow\upharpoonleft_{String}}\underline{\exists}f\mapsto f{:}\mathit{Onto}\triangleleft t\triangleright$

Theorem. $\mathit{Theorem}\triangleleft\rightarrow\upharpoonleft\mathit{String}\triangleright$ is computationally enumerable, i.e., there is a procedure Theorems of type $\mathit{ProvedEnumerator}_{\rightarrow\upharpoonleft_{String}}\triangleleft\mathit{Theorem}\triangleleft\rightarrow\upharpoonleft\mathit{String}\triangleright\triangleright$

Corollary. $\mathit{ProvedTotal}_{\rightarrow\upharpoonleft_{String}}$ is computationally enumerable, i.e., there is a procedure ProvedTotals of type $\mathit{ProvedEnumerator}_{\rightarrow\upharpoonleft_{String}}\triangleleft\mathit{ProvedTotal}_{\rightarrow\upharpoonleft_{String}}\triangleright$.

Definition. Define the procedure Diagonal as follows:

$\text{Diagonal}_{\blacksquare}[i{:}\mathbb{N}] \equiv 1+(\text{ProvedTotals}_{\blacksquare}[i])_{\blacksquare}[i]$

Lemma. Diagonal:$\mathit{ProvedTotal}_{\rightarrow\upharpoonleft_{String}}$

Proof. Suppose i:$\mathbb{N}$. Let f:$\mathit{ProvedTotal}_{\rightarrow\upharpoonleft_{String}}$=ProvedTotals$_{\blacksquare}$[i] and let j:$\mathbb{N}$=f$_{\blacksquare}$[i]. Therefore Diagonal$_{\blacksquare}$[i]=1+j.

Consequently, $\vdash_{\rightarrow\upharpoonleft_{String}}$Diagonal:$\mathit{Total}\triangleleft\mathbb{N}\triangleright$.

Lemma. $\neg$Diagonal:$\mathit{ProvedTotal}_{\rightarrow\upharpoonleft_{String}}$

Proof. Diagonal differs from every $\mathit{ProvedTotal}_{\rightarrow\upharpoonleft_{String}}$ enumerated by ProvedTotals.

Theorem. **The theory $\rightarrow\upharpoonleft\mathit{String}$ is inconsistent** [Church 1934], i.e.,

$$\exists[\underline{\text{Proposition}}_{\rightarrow\upharpoonleft_{String}}\Psi]\ \vdash_{\rightarrow\upharpoonleft_{String}}\Psi\wedge\neg\Psi$$

Proof. Let Ψ=Diagonal:$\mathit{ProvedTotal}_{\rightarrow\upharpoonleft_{String}}$

The upshot is that the theory $\rightarrow$ is algorithmically inexhaustible, i.e., nonalgorithmic creativity will be forever required to develop new $\rightarrow$ axioms abstracted from strings thereby reinforcing the intuition behind [Franzén, 2004]. According to [Church 1934], inconsistency of the theory $\rightarrow\upharpoonleft\mathit{String}$ 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.**

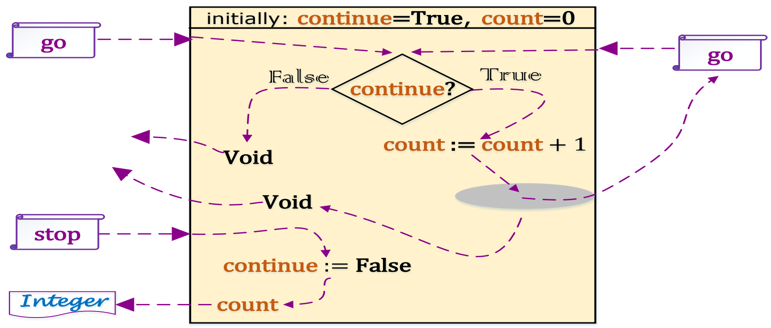
V. Actor Model

[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 λ Calculus, Nondeterministic Turing Machines, or pure Logic Programs

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



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

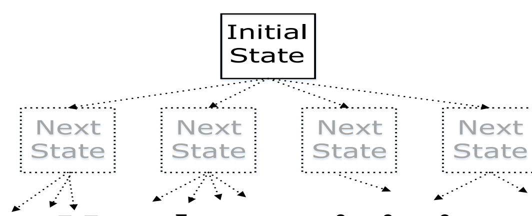
There is an *always-halting* Actor machine that can compute an integer of unbounded size. This is accomplished using an Actor with a variable *count* that is initially 0 and a variable *continue* initially True. The computation is begun by concurrently sending two messages to the Actor machine: a stop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

- When a stop message is received, return *count* and set *continue* to False for the next message received.
- When a go message is received:
  - If *continue* is True, 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 False, return Void.

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 State Change

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

∃[i:N]¬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:

ReadersWriter interface read**[***Query***]** → *ReadResponse*, write**[***Update***]** → *WriteResponse*

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

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

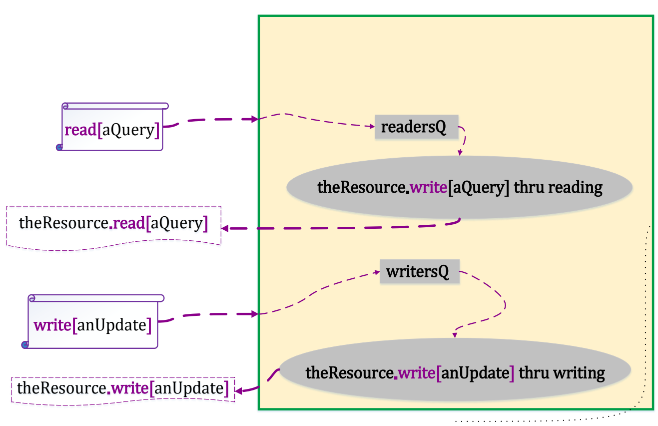
Holes in regions of mutual exclusion

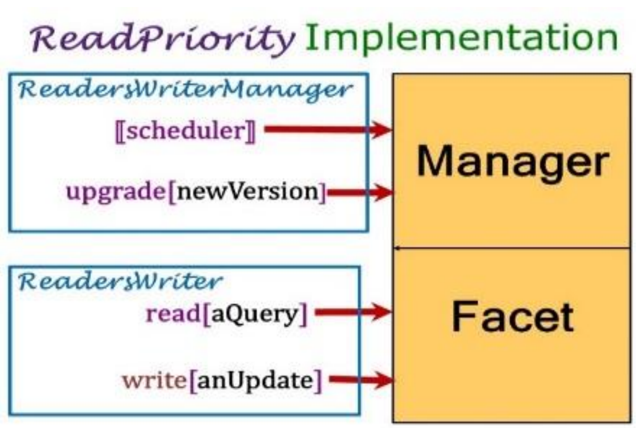
Holes in regions of mutual exclusion (Swiss cheese) [Hewitt and Atkinson 1979; Atkinson 1980] is a generalization of mutual exclusion with the following goals:

- *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.getSchedular means *ReadPriority*.send[x, getSchedular]

   Types manage crypto without requiring programming by application programmers.

Theorem. Readers exclude writers. Suppose $manager_1$ is *ReadPriority*⟦$database_1$⟧**.** After $manger_1$ has sent a write request to $database_1$, 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. *ReadPriority*⟦$database_1$⟧ forwards messages to $database_1$. 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.

Axiomatization of Actors up to a unique isomorphism

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.

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

- Primitive Actors
  - $\forall[i{:}\mathbb{N}]\ i{:}\mathit{Actor}$ // natural numbers are Actors
  - $\forall[x_1,x_2{:}\mathit{Actor}]\ [x_1, x_2]{:}\mathit{Actor}$ // a 2-tuple of Actors is an Actor
- Event ordering
  - $\forall[c{:}\mathit{Com}]\ \exists 1[x_1{:}\mathit{Actor}, c_1{:}\mathit{Com}]\ c{\in}x_1.\mathrm{sent}[c_1]]$ // every communication was sent by an Actor
  - $\forall[c{:}\mathit{Com}]\ \forall[x1, x2{:}\mathit{Actor}]\ \mathrm{Received}_{x1}[c]\wedge\mathrm{Received}_{x2}[c] \Rightarrow x1{=}x2$
    // a communication is received at most once
  - $\forall[x{:}\mathit{Actor}, c{:}\mathit{Com}]\ \mathrm{Initial}_x{\curvearrowright}\mathrm{Received}_x[c]{\curvearrowright}\mathrm{After}_x[c]$
  - $\forall[x{:}\mathit{Actor}, c_1,c_2{:}\mathit{Com}]\ c_1{\neq}c_2 \Rightarrow (\mathrm{Received}_x[c_1]{\curvearrowright}\mathrm{Received}_x[c_2] \vee \mathrm{Received}_x[c_2]{\curvearrowright}\mathrm{Received}_x[c_1])$
  - $\forall[x{:}\mathit{Actor}, c{:}\mathit{Com}]\ \nexists[c_1{:}\mathit{Com}]\ \mathrm{Received}_x[c]{\curvearrowright}\mathrm{Received}_x[c_1]{\curvearrowright}\mathrm{After}_x[c]$
  - $\forall[e_1, e_2{:}\mathit{Event}]\ \mathrm{Finite}[\mathit{Event}{\ni}e{\mapsto}e_1{\curvearrowright}e{\curvearrowright}e_2]$ // There are only finitely many events
    // in $\curvearrowright$ between two events.
- An Actor's behavior change
  - $\forall[x{:}\mathit{Actor}, c{:}\mathit{Com}]\ (\nexists[c_1{:}\mathit{Com}]\ \mathrm{Received}_x[c_1]{\curvearrowright}\mathrm{Received}_x[c])\Rightarrow x_{\mathrm{received}[c]} = x_{\mathrm{initial}}$
  - $\forall[x{:}\mathit{Actor}, c_1,c_2{:}\mathit{Com}]\ (\nexists[c_3{:}\mathit{Com}]\ \mathrm{After}_x[c_1]{\curvearrowright}\mathrm{Received}_x[c_3]{\curvearrowright}\mathrm{Received}_x[c_2])$
    $\Rightarrow x_{\mathrm{received}[c2]} = x_{\blacksquare}\mathrm{after}[c_1]$
  - $\forall[x{:}\mathit{Actor}, c{:}\mathit{Com}]\ \mathrm{Finite}[\mathit{Com}{\ni}s{\mapsto}s{\in}x_{\blacksquare}\mathrm{sent}[c]]$
  - $\forall[x{:}\mathit{Actor}, c{:}\mathit{Com}]\ \mathrm{Info}[x_{\blacksquare}\mathrm{created}[c]] \sqsubseteq \mathrm{Info}[c]\sqcup\mathrm{Info}[\mathrm{Received}_x[c_2]]\sqcup\mathrm{Info}[\mathrm{New}[c]]$
    // New[c] is addresses of Actors created processing c
  - $\forall[x, x_1{:}\mathit{Actor}, c{:}\mathit{Com}]\ x_1{\in}\mathrm{New}[c] \Rightarrow \bot = \mathrm{Info}[x_1]\sqcap(\mathrm{Info}[c] \sqcup \mathrm{Info}[\mathrm{Received}_x[c] \sqcup \mathrm{Info}[\mathrm{New}[c]\text{-}x_1])$
    // info about the address of a newly created Actor does not provide any
    // information about addresses of other current Actors
  - $\forall[x, x_1{:}\mathit{Actor}, c{:}\mathit{Com}, e{:}\mathit{Event}]\ (x_1{\in}\mathrm{New}[c] \wedge e{\curvearrowright}\mathrm{Received}_x[c_2]) \Rightarrow \bot{=}\mathrm{Info}[x_1]\sqcap\mathrm{Info}[e]$
    // info about the address of a newly created Actor does not provide any
    // information about addresses in previous events

○ $\forall$[x_1,x_2:*Actor*, c_1,c_2:*Com*] $c_1 \neq c_2 \wedge x_1 \in$New[$c_1$]$\wedge x_2 \in$New[$c_2$] ⇨ ⊥=Info[$x_1$]⊓Info[$x_2$]
// info about the address of a newly created Actor does not provide any information
// about address of any other newly created Actor
○ $\forall$[x:*Actor*, c:*Com*] Let processing = (Info[c]⊔Info[Received$_x$[c_2]] ⊔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

$\forall$[x:*Actor*, P predicateOn$_{\mathbb{Actor}}$*Behavior*]
(P⟦x_{initial}⟧ ∧ ∀[c:*Com*] P⟦$x_{\text{received[c]}}$⟧
⇨ P⟦x▪after[c]⟧) ⇨ ∀[c:*Com*] P⟦$x_{\text{received[c]}}$⟧ ∧ P⟦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 $\mathbb{Actor}$, i.e., if $\mathcal{M}$ is a type satisfying the axioms for $\mathbb{Actor}$, then there is a unique isomorphism between $\mathcal{M}$ and *TypeIn*◁$\mathbb{Actor}$▷.

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 $\mathbb{T}$ for the model $\mathcal{M}$ for each of the following: Natural Numbers, Real Numbers, Ordinals, Computable Procedures, and Actors. Each theory $\mathbb{T}$ has the following properties:

- $\mathbb{T}$ is uniquely categorical for *Model*◁$\mathbb{T}$▷, i.e., if $\mathcal{X}$ satisfies the axioms of $\mathbb{T}$, then is $\mathcal{X}$ isomorphic to *Model*◁$\mathbb{T}$▷, by a unique isomorphism.
- $\mathbb{T}$ is sound, i.e., $(\vdash_{\mathbb{T}} \Psi) \Rightarrow \Psi$
- For all propositions Ψ of $\mathbb{T}$ and p:*ProofChecker*◁$\mathbb{T}$▷, $\vdash^{\mathrm{p}}_{\mathbb{T}} \Psi$ is computationally decidable.
- $\mathbb{T}$ is **not** compact, i.e., it is not the case that for every inconsistent set of $\mathbb{T}$ 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 $\mathcal{O}$ 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 $\mathit{Boolean}^{\mathcal{O}}$ 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 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, 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 $\mathbb{Actor}$ is based on the way that they are inductively constructed bottom-up.
- *Monsters* [Lakatos 1976] are unwanted elements in models of classical mathematical theories. $\mathbb{Actor}$ 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 $\mathcal{O}$ can be uniquely categorically axiomatized in the theory $\mathbb{O}$ (using induction for the ordinals in a way analogous to induction on $\mathcal{N}$ 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 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."

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 "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
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 $t_1 \rightarrow_1 t_2$, which is the type of *deterministic computable procedures* on t_1 into t_2, but does **not** have ${t_2}^{t_1}$, which is the type of *all* functions on t_1 into t_2. Also, constructive type theory relies on the premise that Ψ is a proposition of theory $\mathbb{T}$ if an only if Ψ is a theorem of $\mathbb{T}$ 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]. **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, 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:

- $\neg\Psi_1, \Psi_1\wedge\Psi_2$:*PropositionOfOrder*◁i▷ where Ψ_1,Ψ_2:*PropositionOfOrder*◁i▷ and i:N_+
- $(x_1=x_2)$:*PropositionOfOrder*◁1▷ where x_1,x_2:*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_+
- $(\Psi_1\vdash\Psi_2)$:*PropositionOfOrder*◁i▷ where i:N_+ and Ψ_1,Ψ_2:*PropositionOfOrder*◁i▷
- $(\Psi_1 \underset{\mathbb{T}}{\overset{p}{\vdash}} \Psi_2)$:*PropositionOfOrder*◁i▷ where p:*Term*◁*ProofChecker*▷, $\mathbb{T}$:*Theory*, Ψ_1,Ψ_2:*PropositionOfOrder*◁i▷ and i:N_+
- ⌊s⌋:*PropositionOfOrder*◁i▷ is abstraction of s where s:*String*◁*PropositionOfOrder*◁i▷▷ with no free variables and i:N_+
- ⌈Ψ⌉:*String*◁*PropositionOfOrder*◁i▷▷ is quotation of Ψ where Ψ:*PropositionOfOrder*◁i▷↾*String*, and i:N_+.

Procedural *Expression* is a discrimination of the following:

- $x{:}\mathit{Expression}\triangleleft t\triangleright$ where $x{:}\mathit{Constant}\triangleleft t\triangleright$ and t is a type
- $x{:}\mathit{Expression}\triangleleft t\triangleright$ where $x{:}\mathit{Identifier}\triangleleft t\triangleright$ and t is a type
- $[e_1, e_2]{:}\mathit{Expression}\triangleleft [t_1, t_2]\triangleright$ where $e_1{:}\mathit{Expression}\triangleleft t_1\triangleright$, $e_2{:}\mathit{Expression}\triangleleft t_2\triangleright$, and t_1 and t_2 are types
- $(e_1$ **??** <u>True</u> <u>then</u> e_2 , <u>False</u> <u>then</u> $e_3){:}\mathit{Expression}\triangleleft t\triangleright$ where $e_1{:}\mathit{Expression}\triangleleft \mathit{Boolean}\triangleright$, $e_2, e_3{:}\mathit{Expression}\triangleleft t\triangleright$ and t is a type
- $(\lambda[x{:}t_1]\ y){:}\mathit{Expression}\triangleleft t_1\rightarrow t_2\triangleright$ where $x{:}\mathit{Identifier}\triangleleft t_1\triangleright$, $y{:}\mathit{Expression}\triangleleft t_2\triangleright$ and t_1 and t_2 are types
- $x_{\blacksquare}m{:}\mathit{Expression}\triangleleft t_2\triangleright$ where $m{:}\mathit{Expression}\triangleleft t_1\triangleright$, x is an Actor with a message handler with signature of type $\mathit{Expression}\triangleleft t_1\rightarrow t_2\triangleright$, and t_1 and t_2 are types
- $I[\![x_1, \ldots, x_n]\!]{:}\mathit{Expression}\triangleleft I\triangleright$ where I is an Actor implementation and $x_1, \ldots, x_n$ are expressions.
- $\lfloor s\rfloor{:}\mathit{Expression}\triangleleft t\triangleright$ is abstraction of s where $s{:}\mathit{String}\triangleleft \mathit{Expression}\triangleleft t\triangleright\triangleright$ with no free variables and t is a type
- $\lceil x\rceil{:}\mathit{String}\triangleleft \mathit{Expression}\triangleleft t\triangleright\triangleright$ is quotation of x where $x{:}\mathit{Expression}\triangleleft t\triangleright\upharpoonleft\mathit{String}$, $i{:}\mathbb{N}_+$ and t is a type.

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

- $x{:}\mathit{Term}\triangleleft t\triangleright$ where $x{:}\mathit{Constant}\triangleleft t\triangleright$ and t is a type
- $x{:}\mathit{Term}\triangleleft t\triangleright$ where $x{:}\mathit{Variable}\triangleleft t\triangleright$ and t is a type
- $[x_1, x_2]{:}\mathit{Term}\triangleleft [t_1, t_2]\triangleright$ where $x_1{:}\mathit{Term}\triangleleft t_1\triangleright$, $x_2{:}\mathit{Term}\triangleleft t_2\triangleright$, and t_1 and t_2 are types
- $(x_1$ **??** <u>True</u> <u>then</u> x_2 , <u>False</u> <u>then</u> $x_3){:}\mathit{Term}\triangleleft t\triangleright$ where $x_1{:}\mathit{Term}\triangleleft \mathit{Boolean}\triangleright$, $x_2, x_3{:}\mathit{Term}\triangleleft t\triangleright$ and t is a type
- $([x{:}t_1]\mapsto y){:}\mathit{Term}\triangleleft {t_2}^{t_1}\triangleright$ where $x{:}\mathit{Variable}\triangleleft t_1\triangleright$, $y{:}\mathit{Term}\triangleleft t_2\triangleright$ and t_1 and t_2 are types
- $f[x]{:}\mathit{Term}\triangleleft t_2\triangleright$ where $f{:}\mathit{Term}\triangleleft {t_2}^{t_1}\triangleright$, $x{:}\mathit{Term}\triangleleft t_1\triangleright$, and t_1 and t_2 are types
- $\lfloor s\rfloor{:}\mathit{Term}\triangleleft t\triangleright$ is abstraction of s where $s{:}\mathit{String}\triangleleft \mathit{Term}\triangleleft t\triangleright\triangleright$ with no free variables and t is a type
- $\lceil x\rceil{:}\mathit{String}\triangleleft \mathit{Term}\triangleleft t\triangleright\triangleright$ is quotation of x where $x{:}\mathit{Term}\triangleleft t\triangleright\upharpoonleft\mathit{String}$, $i{:}\mathbb{N}_+$ 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/Girad [Burali-Forti 1897, Girard 1972, Coquand 1986]
Although each ordinal α can be strictly embedded as a well-founded order in the ordinals $\mathcal{O}$ and $\alpha =_{\mathcal{O}} \mathcal{O}\ni\beta\mapsto\beta{:}\alpha$ as Ordinals, $\neg\mathcal{O}{:}(\mathcal{O}\ni\beta\mapsto\beta{:}\mathcal{O})$ 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]

- 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 $P\mapsto\neg P[\![P]\!]$ has **no** fixed point because $\neg P[\![P]\!]$ 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 $\Psi\mapsto\nvdash\Psi$ does **not** have a fixed point (contra [Gödel 1931]) because the order of $\nvdash\Psi$ is greater than the order of Ψ since Ψ is a propositional variable.

Curry [Curry 1941]
Curry's Paradox is blocked because the mapping $p\mapsto(p\Rightarrow\Psi)$ 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 $P\mapsto(\forall[i,j>i:\mathcal{N}]\ \neg P[\![j]\!])$ does **not** have a fixed point because the order of $\forall[i,j>i:\mathcal{N}]\ \neg P[\![j]\!]$ is one great 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$\triangleleft$i$\triangleright[\![s, k]\!]$ meaning that the string s characterizes the integer k as follows where i:$\mathcal{N}_+$:

- $\mathcal{Berry}\triangleleft i\triangleright \equiv (\mathcal{Term}\triangleleft\mathcal{PropositionofOrder}\triangleleft i\triangleright^{\mathcal{N}})\upharpoonright\mathcal{String}$
- Characterize$\triangleleft$i$\triangleright[\![s{:}\mathcal{Berry}\triangleleft i\triangleright,\ k{:}\mathcal{N}]\!] \equiv \forall[x{:}\mathcal{N}]\lfloor \mathbf{s}\rfloor[\![x]\!] \Leftrightarrow x=k$

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:

BerryString:$\mathcal{Berry}\triangleleft i+1\triangleright \equiv$ "$[j{:}\mathcal{N}]\mapsto\forall[s{:}\mathcal{PropositionOfOrder}\triangleleft i\triangleright\upharpoonright\mathcal{String}]$
Length[s]$<100 \Rightarrow \neg$Characterize$\triangleleft$i$\triangleright[\![s, j]\!]$"

Note that

- Length[BerryString]<100.
- $\mathcal{Berry}\triangleleft i\triangleright\underline{\exists}s\mapsto$Length[s]$<100$ is finite.
- Therefore, $\mathcal{BerryNumber}$ is finite where
  $\mathcal{BerryNumber} \equiv \mathcal{N}_+\underline{\exists}j\mapsto\exists[s{:}\mathcal{Berry}\triangleleft i\triangleright]$Length[s]$<100 \wedge$ Characterize$\triangleleft$i$\triangleright[\![s, j]\!]$
- $\exists[i{:}\mathcal{N}_+]\ i{:}\mathcal{BerryNumber}$ because is $\mathcal{N}_+$ is infinite.
- LeastBerry $\equiv$ Least[$\mathcal{BerryNumber}$]
- $\lfloor$BerryString$\rfloor[\![$LeastBerry$]\!] = \forall[s{:}\mathcal{Berry}\triangleleft i\triangleright]$ Length[s]$<100 \Rightarrow \neg$Characterize$\triangleleft$i$\triangleright[\![$s, LeastBerry$]\!]$

However BerryString:$\mathcal{Berry}\triangleleft i+1\triangleright$ **cannot be substituted** for s:$\mathcal{Berry}\triangleleft i\triangleright$. Consequently, **the Berry Paradox as follows does not hold**:

$\lfloor$BerryString$\rfloor[\![$LeastBerry$]\!] \Leftrightarrow \neg$Characterize$\triangleleft$i$\triangleright[\![$BerryString, LeastBerry$]\!]$

APPENDIX: ORDINALS AND NATURAL NUMBERS

Theory of Natural Numbers

The mathematical theory ℕ that axiomatises the Natural Numbers N 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
- $\nexists[i{:}N]\ +_1[i]=0$ // 0 is not a successor
- $\forall[i,j{:}N]\ +_1[i]=+_1[j] \Rightarrow i=j$ // $+_1$ is 1 to 1

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

$$\forall[P\ \underline{\text{predicateOn}}\ N]\ (P[\![0]\!] \land \forall[i{:}N]\ P[\![j]\!]\Rightarrow P[\![+_1[j]]\!]) \Rightarrow \forall[j{:}N]\ P[\![j]\!]$$

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

- Define by induction on $TypeIn\triangleleft\mathbb{N}\triangleright$]
  - $I[N] \equiv M$
  - $I[\,[t_1,t_2]\,] \equiv [I[t_1], I[t_2]\,]_M$
  - $I[t2^{t1}] \equiv I[t2]^{I[t1]}$
  - $I[t_1\rightarrow t_2] \equiv I[t2]\rightarrow I[t2]$
  - $\forall[P\ \underline{\text{predicateOn}}_{\mathbb{N}}\ TypeIn\triangleleft\mathbb{N}\triangleright]\ I[t\,\underline{\ni}P] \equiv I[t]\underline{\ni}I[P]$
- Define by induction on $TypeIn\triangleleft\mathbb{N}\triangleright$]
  - Define by induction on N
    - $I[0] \equiv 0_M$
    - $I[+_1[j]] \equiv +_1^M[I[j]]$
  - if $x{:}\,[t_1,t_2]$, then $I[x] \equiv [I[1^{st}[x]], I[2^{nd}[x]\,]]_M$
  - if $x{:}t2^{t1}$, then $I[x] \equiv y{:}I[t_2]\mapsto I[x[I^{-1}[y]]]$ // inductive hypothesis for I on t_2

I is a unique isomorphism because of the following;

- I is defined on $TypeIn\triangleleft\mathbb{N}\triangleright$
- I is 1-1
- I is onto M
- I is a homomorphism
- I^{-1} is a homomorphism
- If g is an isomorphism of $Model\triangleleft\mathbb{N}\triangleright$ with M, then g=I

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

Theory of Ordinals

The theory 𝕆 that axiomatises the ordinals O has the following axioms in addition to the axioms for the theory ℕ:

- $0{:}O$ // 0 is an ordinal
- $\nexists[\alpha{:}O]\ \alpha{:}0$ // 0 has no predecessor
- $+_1{:}1to1\triangleleft O,O\triangleright$ // $+_1[\alpha]=\alpha+1$
- $\forall[\alpha{:}O]\ \alpha{:}+_1[\alpha]$ // $\alpha{:}\beta \Leftrightarrow \alpha<\beta$
- $\forall[\alpha,\beta,\gamma{:}O]\ \alpha{:}\beta\land\beta{:}\gamma \Rightarrow \alpha{:}\gamma$
- $\forall[\alpha{:}O]\ \nexists[\beta{:}O]\ \alpha{:}\beta \land \beta{:}+_1[\alpha]$
- $\forall[\alpha{:}O,\ f{:}Nondecreasing\triangleleft O\triangleright]\ \uplus_\alpha f{:}O$ // $\uplus_\alpha f$ is limit of range of f on α

- $\forall[\alpha{:}\mathcal{O},\ f{:}\mathit{Nondecreasing}\triangleleft\mathcal{O}\triangleright]\ \uplus_0 f=0$
- $\forall[\alpha{:}\mathcal{O},\ f{:}\mathit{Nondecreasing}\triangleleft\mathcal{O}\triangleright]\ \uplus_{\alpha+1} f=f[\alpha]+1$
- $\forall[\alpha,\beta<\alpha{:}\mathcal{O};\ f{:}\mathit{Nondecreasing}\triangleleft\mathcal{O}\triangleright]\ f[\beta]\leqq\uplus_\alpha f$ // $\uplus_\alpha f$ includes all of range of f on α
- $\forall[\alpha,\beta{:}\mathcal{O};\ f{:}\mathit{Nondecreasing}\triangleleft\mathcal{O}\triangleright]\ (\forall[\gamma<\alpha]\ f[\gamma]\leqq\beta)\Rightarrow(\uplus_\alpha f)\leqq\beta$
  // $\uplus_\alpha f$ is minimum of range of f on α
- $\forall[\alpha{\cdot}\mathcal{O}]\ \omega_\alpha{:}\mathcal{O}$
- $\omega_0=\mathcal{N}$
- $\forall[\alpha{:}\mathcal{O}]\ \mathrm{1to1}[\omega_{\alpha+1}, \mathit{Boolean}^{\omega_\alpha}]$ // jump in cardinality
- $\forall[\alpha,\gamma{:}\mathcal{O}]\ \mathrm{1to1}[\gamma, \mathit{Boolean}^{\omega_\alpha}]\Rightarrow\omega_{\alpha+1}\leqq\gamma$ // minimal jump in cardinality
- $\forall[\alpha{:}\mathcal{O}]\ (\nexists[\beta{:}\mathcal{O}]\ \alpha=+_1[\beta])\Rightarrow\omega_\alpha=\uplus_\alpha(\beta\mapsto\omega_\beta)$
  // ω of limit ordinal is limit of ωs of all lesser ordinals

In addition, the theory $\mathbb{O}$ has the following induction axiom, *which has uncountable instances*:

$$\forall[P\ \underline{\mathrm{predicateOn}}_{\mathbb{O}}\ \mathcal{O}]\ (P[\![0]\!]\wedge\forall[\alpha{:}\mathcal{O}]\ \forall[\beta<\alpha{:}\mathcal{O}]\ P[\![\beta]\!]\Rightarrow P[\![\alpha]\!]]\!])\Rightarrow\forall[\alpha{:}\mathcal{O}]\ P[\![\alpha]\!]$$

Theorem.: $\mathcal{O}$ is well-ordered by <, i.e. $\nexists[f{:}\mathcal{O}^{\mathcal{N}}]\ \forall[i{:}\mathcal{N}]\ f[i+1]<f[i]$

Theorem.: If $\mathcal{M}$ is a type satisfying the axioms of the theory $\mathbb{O}$, then there is a unique isomorphism $I{:}\mathcal{M}^{\mathit{Model}\triangleleft\mathbb{O}\triangleright}$ defined as follows:

- Define by induction on $\mathit{TypeIn}\triangleleft\mathbb{O}\triangleright$
  - $I[\mathcal{O}]\equiv\mathcal{M}$
  - $I[\,[t_1,t_2]\,]\equiv[I[t_1], I[t_2]\,]_{\mathcal{M}}$
  - $I[t2^{t1}]\equiv I[t2]^{I[t1]}$
  - $\forall[P\ \underline{\mathrm{predicateOn}}_{\mathbb{O}}\ \mathit{TypeIn}\triangleleft\mathbb{O}\triangleright,\ t{:}\mathit{TypeIn}\triangleleft\mathbb{O}\triangleright]\ \ I[t\,\underline{\ni}P]\equiv I[t]\underline{\ni}I[P]$
- Define by induction on $\mathit{TypeIn}\triangleleft\mathbb{O}\triangleright$
  - Define by induction on $\mathcal{O}$
    - $I[0]\equiv 0_{\mathcal{M}}$
    - $I[+_1[\alpha]]\equiv +_1^{M}[I[\alpha]]$
    - $I[\omega_\alpha]\equiv\omega_{I[\alpha]}^{M}$
    - otherwise $I[\uplus_\alpha f]\equiv\uplus_{I[\alpha]}^{M} I[f]$
  - if $x{:}[t_1,t_2]$, then $I[x]\equiv[I[1^{st}[x]], I[2^{nd}[x]\,]]_{\mathcal{M}}$
  - if $x{:}t2^{t1}$, then $I[x]\equiv y{:}I[t_2]\mapsto I[x[I^{-1}[y]]]$ // inductive hypothesis for I on t_2

References

M. Amborn. *Facet-Oriented Program Design*. LiTH-IDA-EX–04/047–SE Linkőpings Universitet. 2004.

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.

N. Bourbaki. *Elements of Mathematics* Springer. 1939-2016.

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

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.
T. Coquand. *An Analysis of Girard's Paradox* INRIA. Report 531. May 1986.
T. Coquand and G. Huet. *The calculus of constructions.* Technical Report 530, INRIA, Centre de Rocquencourt, 1986.
E. Crahen. *Facet: A pattern for dynamic interfaces*. CSE Dept. SUNY at Buffalo. July 22, 2002.
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
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* These d'Etat. Paris VII. 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.
M. Gordon, R. Milner and C. Wadsworth. (1979) *Edinburgh LCF: A Mechanised Logic of Computation* Lecture Notes in Computer Science. Vol. 78. Springer-Verlag. 1979.
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.
J. Harrison. *HOL Light Tutorial* Intel Corporation. January 14, 2017.
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: Faster Response Time and Better Information Integration Than Datacenters of Competing Companies for International Commerce and Representative Government* 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. *Inference Robustness* 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.
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.
P. Martin-Löf. *A Theory of Types* Stockholm University. Technical Report 71–3. 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.
G. Plotkin. *A powerdomain construction* SIAM Journal of Computing. September 1976.
N. Perlroth, D. Sanger and S. Shane. *How Chinese Spies Got the N.S.A.'s Hacking Tools, and Used Them for Attacks*. New York Times. May 6, 2019.
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.
B. Russell. *Logical positivism* Revue internationale de philosophie. Vol. 4. 1950.
S. G. Shanker. *Wittgenstein's Remarks of the Significance of Gödel's Theorem* Godel's Theorem in Focus. Croom Helm. 1988.
R. Sobers. *60 Must-Know Cybersecurity Statistics for 2019*. Varonis. April 17, 2019.
A. Turing. *On Computable Numbers, with an Application to the Entscheidungsproblem* Proceedings of the London Mathematical Society. 2. 42. 1936.
A. Urquhart. *Russell and Gödel* Bulletin of Symbolic Logic. Volume 22, Number 4, December 2016.
A. Wiles. *Modular elliptic curves and Fermat's Last Theorem* Annals of Mathematics. 141 (3). 1995.
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* Inference Robustness, Studies in Logic. 2019.
S. Yablo. *Truth and reflection* Journal of Philosophical Logic. 14 (2). 1985.