Computer Science Must Rely on Strongly-Typed Actors and Theories for Cybersecurity
Carl Hewitt

To cite this version:
Carl Hewitt. Computer Science Must Rely on Strongly-Typed Actors and Theories for Cybersecurity. Inference Robustness, In press. hal-02165430v2

HAL Id: hal-02165430
https://hal.archives-ouvertes.fr/hal-02165430v2
Submitted on 10 Jul 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
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 [Hewitt 2019]) up to a unique isomorphism. Strongly-typed Actors provide foundations for tremendous improvements in cyberdefense.


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

† 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/](https://professorhewitt.blogspot.com/)
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 types and theories of these structures.

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 \( \iff \neg \text{I'mFalse} \), 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 FOUNDATIONS

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., \( \vdash_{\text{Russell}} \neg \text{RussellUnprovable} \)] where \( P \equiv \text{RussellUnprovable} \) then by this proof I have proved \( P \) [i.e., \( \vdash_{\text{Russell}} P \) because \( P \equiv \text{RussellUnprovable} \)]. Now if this proof were one in Russell’s system [i.e., \( \vdash_{\text{Russell}} \neg \text{RussellUnprovable} \)] — I should in this case have proved at once that it belonged [i.e., \( \vdash_{\text{Russell}} P \)] and did not belong [i.e., \( \vdash_{\text{Russell}} \neg P \) because \( \neg P \leftrightarrow \vdash_{\text{Russell}} P \)] to Russell’s system. But there is a contradiction here! [i.e., \( \vdash_{\text{Russell}} P \) and \( \vdash_{\text{Russell}} \neg P \)] ...

[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 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[Russell] \( \iff \vdash_{\text{Russell}} \neg \text{RussellUnprovable} \) where \( P \equiv \text{RussellUnprovable} \) and Consistent[Russell] if an only if there is no proposition \( \Psi \) such that \( \vdash_{\text{Russell}} \Psi \land \neg \Psi \). However, Wittgenstein was understandably taking it as a given that Russell is consistent because it formalized standard mathematical practice and had been designed to block known paradoxes (such as The Liar) using orders on propositions. Consequently, Wittgenstein elided the result in [Gödel 1931] to \( \vdash_{\text{Russell}} \neg \text{RussellUnprovable} \). His point was that Russell is consistent provided that the proposition \( \vdash_{\text{Russell}} \neg \text{RussellUnprovable} \) is not added to Russell. Wittgenstein was justified in assuming consistency of Russell because the natural 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 $\Psi \mapsto \lnot \Psi$ does not have a fixed point because the order of $\lnot \Psi$ 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 tests), 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 $T$ that can formalize its own provability has a model $M$ with a Gödelian “monster” element proposition $\Gamma$ that proves $T$ inconsistent (i.e. $\vdash_M \neg T \Gamma \land \neg \Gamma$) by the following proof:
    
    According to [Gödel 1931], $\lnot T \text{Consistent}[T]$ and consequently because of the 1st-order model “completeness” theorem [Gödel 1930] there must be some model $M$ of $T$ in which Consistent$[T]$ 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 use.

- **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, $0 : \mathbb{N}$ expresses that 0 is of type $\mathbb{N}$, which is the type of a natural number. Types are *intensional*, i.e., if $x : t_1 \equiv x : t_2$ 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 $\text{TypeOf}<t>$. 

• $t_2^t$ is type of all functions from $t_3$ 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 $t_1 \rightarrow_n^1 t_2$ 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 $\Psi$ means $\exists [i: N_+] \Psi: PropositionOfOrder<1>$

  ○ $P$ predicateOn $t$ means $\exists [i: N_+].P: PropositionOfOrder<i>^t$

• $t^P$ 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 $f:t_2^t$ is $t_2 \exists y \rightarrow \exists [x:t_1] y = f[x]$.

• $TypeOf<\tau>$ is the type of the type $\tau$. For example, $N:TypeOf<N>$ (cf. [Martin-Löf 1998])

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. FOUNDATIONAL THEORY OF DIGITAL COMPUTATION

Cybersecurity requires that fundamental mathematical structures in Computer Science must be precisely defined. This section shows how to axiomatize classical nondeterministic computable procedures up to a unique isomorphism including computable nondeterministic procedures that cannot be implemented using a Lambda Expression [Church 1932] or Turing Machine [Turing 1936].

Computation that cannot be done by λ Calculus, Nondeterministic Turing Machines, or pure Logic Programs [Church 1931] 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, which inspired the development of the Actor Model in 1972 to characterize all digital computation.

Theorem. An Actor machine can perform computations that a no λ expression, nondeterministic Turing

Resend go message until stop message received
Machine or pure Logic Program can implement because there is an *always-halting* Actor machine that can compute an integer of unbounded size (cf. [Clinger 1981]) This can be 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, Parallel Program Schemata [Karp and Miller 1967] or pure Logic Program [Hewitt 1969] 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."

A classification should be developed for nondeterministic computable procedures that are not Church/Turing computable, e.g., in terms of messages sent.

An Actor machine can be thousands of times faster than any corresponding pure Logic Program or parallel nondeterministic λ expression. (cf. [Kahn 1979, Kornfeld 1981, Hewitt and Woods assisted by Spurr 2019]). Since the time of this early work, Actors have grown to be one of the most important paradigms in computing [Hewitt and Woods assisted by Spurr 2019, Hoare 2018, Milner 1993].

**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 \( \exists [i:]N \rightarrow Output[i] \) is inconsistent but every finite subset \( i \) 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 non-interactive conceptions of [Church 1931] and [Turing 1936]. Although 1st-order propositions can be useful (e.g. in testing 1st-order propositions for satisfiability), interactive concurrency 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:

```plaintext
ReadersWriter interface read[Query] → ReadResponse, // results of read query on database
                        write[Update] → WriteResponse // perform update on database
```

[Type here]
Below is an implementation of `ReadersWriter` that does not allow concurrency in aDatabase:

```plaintext
OneAtATime[aDatabase:ReadersWriter] implements ReadersWriter

read[aQuery] ↦ // read[aQuery] message received
    aDatabase • read[aQuery] // forward query while not receiving any messages until finished
write[anUpdate] ↦ // write[anUpdate] request received
    aDatabase • read[aQuery] // forward request while not receiving any messages until finished
```

Although, the `OneAtATime` implementation of `ReadersWriter` excludes all other activity when there is writing activity in aDatabase, it does not allow multiple reading activities when there is no writing activity.

### Holes in regions of mutual exclusion

In order to implement more general scheduling policies, a region of mutual exclusion can have holes (cf. [Atkinson 1980, Brinch Hansen 1996, Hewitt and Atkinson 1979, Hoare 1974]).

Below is an implementation `SchedulerManager` with facets `writePriority`, and `readPriority` that both implement `ReadersWriter`:

```plaintext
SchedulerManager[aDatabase:ReadersWriter] actor implements Scheduler

Use[Scheduler], // use Scheduler for this implementation

Contained[ ] // contained locals cannot be sent to other Actors
    writersQ ← New Suspended[], // FIFO of suspended writers
    readersQ ← New Suspended[], // FIFO of suspended readers
    reading ← New Pending[], // set of active readers
    writing ← New Pending[1], // at most 1 member in set of active writers

    // Invariant: Nonempty[writing] ⇒ IsEmpty[reading]
getWritePriority ↦ As writePriority, // get writePriority facet
getReadPriority ↦ As readPriority, // get readPriority facet
upgrade[newVersion] ↦ /* upgrade request received */ { // upgrade in place to a new version
    CancellAll Scheduler // CancellAll implementation from Scheduler
    (writersQ, writersQ, reading, writing);
    // Require: AllEmpty[readersQ, writersQ, reading, writing]
    Become newVersion
}
```

![Diagram of SchedulerManager actor with facets](image)
Different scheduling policies can be implemented by facets of SchedulerManager (cf. [Amborn 2004, Crahen 2002]) as shown below in this section.

A write priority policy can provide readers with more recent information (but with potentially less throughput) than a read priority policy. Below is an implementation of the writePriority facet of SchedulerManager:

```
writePriority facet SchedulerManager implements ReadersWriter
    read[aQuery] ↦ /* read query received */ {
        Enqueue readersQ when SomeNonempty(writing, writersQ, readersQ);
        // Require: IsEmpty[writing]
        aDatabase.read[aQuery] thru reading // forward query while recording pending in reading
            afterwrd // Require: IsEmpty[writing]
                permit writersQ when IsEmpty(reading) else (readersQ when IsEmpty(writersQ))},
    }
```

Below is an implementation of doWrite which is an implementation of SchedulerManager that has been separated out for convenient reuse in different facets:

```
doWrite implementation SchedulerManager
    write`ReadersWriter[anUpdate] ↦ /* write request of ReadersWriter received */ {
        Enqueue writersQ when SomeNonempty(reading, writing, readersQ, writersQ);
        // Require: AllEmpty[writing, reading]
        aDatabase.write[anUpdate] thru writing // forward request while recording pending in writing
            afterwrd // Require: AllEmpty[writing, reading]
                permit readersQ else writersQ}
```

A read priority policy can have more throughput (but readers can potentially get less recent information) than a write priority policy. Below is an implementation of the readPriority facet of SchedulerManager:

```
readPriority facet SchedulerManager implements ReadersWriter
    read[aQuery] ↦ /* read query received */ {
        Enqueue readersQ when SomeNonempty(writing, writersQ, readersQ);
        // Require: IsEmpty[writing]
        aDatabase.read[aQuery] thru reading // forward query while recording pending in reading
            permit readersQ // implements read priority policy
                afterwrd // Require: IsEmpty[writing]
                    permit writersQ when IsEmpty(reading) else (readersQ when IsEmpty(writersQ))],
    write`doWrite // write method from doWrite implementation of SchedulerManager
```

Note:

1. At most one activity is allowed to execute in the region of mutual exclusion of an Actor.
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. Instead control moves through an Actor in accord with the scheduling performed by the Actor in response to communications received.
3. An implementation, e.g. SchedulerManager, differs from a class [Dahl and Nygaard 1967] as follows:
   - An implementation can use multiple other implementations (thereby avoiding having to copy and paste code) using qualified names to prevent ambiguity, i.e., not relying on default selections in ambiguous cases as in C++ [ISO 2017].
   - An implementation cannot be subclassed in order to prevent impersonation by other types.
4. An invariant for an Actor must hold when it is created and when entering/leaving a continuous section of a region of mutual exclusion.

5. Strong types are the foundation of Actor communication. For example, if x is of type \texttt{ReadPriority}, then \texttt{x.getScheduler} means \texttt{ReadPriority} send[getScheduler to x]

Types manage crypto without requiring programming by application programmers.

\textbf{Theorem.} Readers exclude writers from a database. Suppose manager$_1$ is \texttt{New ReadPriority}[database$_1$]. After manager$_1$ has sent a write request to database$_1$, it will not send another request to until it has received a response because the invariant

\texttt{Nonempty[writing]⇒IsEmpty[reading]} holds as follows:

- The invariant holds when a \texttt{ReadPriority} implementation is created.
- If the invariant holds in a \texttt{ReadPriority} implementation when a communication is received, then it holds when has been processed.

\textbf{Theorem.} \texttt{New ReadPriority}[database$_1$] forwards messages to database$_1$. Starvation of activities suspended in readersQ and writersQ as is prevented in a \texttt{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.

\textbf{Axiomatization of Actors up to a unique isomorphism}

Let \texttt{x}[e] be the behavior of Actor \texttt{x} at local event \texttt{e}, \texttt{Com} be the type for a communication, and \texttt{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 relationship on events and Info[\texttt{x}] is the information in the Actor addresses of \texttt{x}:

- **Primitive Actors**
  - $\forall[i: \mathbb{N}]: \texttt{i:Actor}$ // 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

- **Actor behavior**
  - $\forall[x: \texttt{Actor}, c: \texttt{Com}]: \texttt{(\exists[c_1: \texttt{Com}] \hspace{1em} Received}_x[c_1] \Rightarrow Received_x[c]) \Rightarrow x_{\text{received}}[c] = x_{\text{initial}}$
  - $\forall[x: \texttt{Actor}, c_1, c_2: \texttt{Com}]: \texttt{(\exists[c_3: \texttt{Com}] \hspace{1em} After}_x[c_1] \Rightarrow Received}_x[c_3] \Rightarrow Received}_x[c_2])$
    - $x_{\text{received}}[c_2] = x_{\text{after}}[c_1]$
  - $\forall[x: \texttt{Actor}, c: \texttt{Com}]: \texttt{Finite}[	exttt{Com} \Rightarrow s \mapsto s \cdot x_{\text{sent}}[c]]$ // only finitely messages are sent while processing a communication

- **Events**
  - $\forall[c: \texttt{Com}]: \exists[e: \texttt{Event}]: \texttt{Sent}_c[e]$ // a communication was sent in exactly 1 event
  - $\forall[c: \texttt{Com}, e_1, e_2: \texttt{Event}, c: \texttt{Com}]: e_1: \texttt{Received}_c[c] \land e_2: \texttt{Received}_c[c] \Rightarrow e_1 = e_2$
    // a communication is received at most once
  - $\forall[x: \texttt{Actor}, c: \texttt{Com}]: x: \texttt{Created}_c ⇐ \texttt{Initial}_x = \texttt{Creation}_x$
    // initial event of a created Actor is its creation event
∀[x:Actor, c:Com] Initial ⊢ Received x[c] ⊢ After x[c]

∀[x:Actor, c1,c2:Com] c1≠c2 ⇒ (Received x[c1] ⊢ Received x[c2] V Received x[c2] ⊢ Received x[c1])

// an Actor imposes an order in which communications are received

∀[x:Actor, c:Com] ∃[c1:Com'] Received x[c] ⊢ Received x[c1] ⊢ After x[c]

// no communication is received while an Actor is in its region of mutual exclusion

∀[c:Com] ∃[e:Event] Sender[c] ⊢ e ⊢ Receiver[c]

// there is no event between sending a receiving a communication

∀[e1, e2, e3:Event] e1 ⊢ e2 ∧ e2 ⊢ e3 ⇒ e1 ⊢ e3  // ≺ is transitive

Event Induction:

∀[P predicateOn Actor Event]
  (∀[x:Actor] P[Initial x]  // If P holds for every initial event of every Actor and
   ∃∀[x:Actor, c:Com]  // For every Actor and communication
   P[Received x[c]]  // If P holds for a received event of the Actor, then
   ⇒ ((∀[e:Activated <c>] P[e]) ∧ ∀[e:Received NextAfter <c>] P[e])  // P holds for subsequent immediately activated events of the communication
   // and P holds for any immediately subsequently received event of the Actor
   ⇒ ∀[e:Event] P[e]  // then P holds for every event

• Bits of an address cannot be inferred without being communicated from the creator of the address (cf. [Hewitt and Baker 1977]):

  ∀[e, e1:Event, c:Com, x:Actor] e1:Creation <c> ∧ e ⊢ Received x[c] ⇒ ⊥= Info[e1] ∩ Info[e]

  // info about the address of a newly created Actor does not provide any
  // information about addresses in previous events

  ∀[e1,e2:Event, c1,c2:Com] c1≠c2 ∧ e1:Creation <c1> ∧ e2:Creation <c2> ⇒ ⊥= Info[e1] ∩ Info[e2]

  // info about the address of a newly created Actor does not provide any information
  // about address of newly created Actor by a different communication

  ∀[e1,e2:Event, c:Com] e1≠e2 ∧ e1:Creation <c> ∧ e2:Creation <c> ⇒ ⊥= Info[e1] ∩ Info[e2]

  // info about the address of a newly created Actor does not provide any information
  // about address of any other newly created Actor of the same communication

  ∀[x:Actor, c:Com] Info[x after[c]] ⊪ Info[Received x[c]] ∪ Info[x created[c]]

  // info about addresses in x after processing c is contained in the information when
  // c was received together with info created as a result of processing 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 from the receiver of the request.

Theorem. Actor Induction, i.e.,

∀[x:Actor, P predicateOn Actor Behavior]
  (P[Initial x] ∧ ∀[c:Com] P[x received[c]] ⇒ P[x after[c]]) ⇒ ∀[c:Com] P[x received[c]] ∧ P[x after[c]]

Proof. Follows immediately from Actor Event Induction axiom.

Theorem. ≺ is asymmetric, i.e., ∀[e1, e2:Event] e1 ≺ e2 ⇒ ¬e2 ≺ e1

Theorem. Every event is a receipt or activated by unique communication, i.e.,

∀[e:Event] ∃1[c:Com] e:Received <c> V e:Activated <c>

Theorem. Digital computation requires time, i.e., ∀[e1,e2:Event] Finite[Event] e1⇒e2 ∧ e1≃ e2

Proof. Follows from Actor Event Induction on events that follow e1 in ≺ ordering.

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>.

Proof. Follows from Actor Event Induction.
**Thesis.** Any digital system can be directly modeled and implemented using Actors.

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

**Actor Program Expressions**

$\text{Eval} < t_\triangleright : [\text{Expression} < t_\triangleright \text{ using Environment}] > t$ is a procedure [McCarthy et. al. 1962] that corresponds to a universal Turing machine [Turing 1936] as follows:

- $\text{Eval} < \text{Expression} < t_\triangleright > x ] \equiv \text{Eval} < t_\triangleright [x \text{ using EmptyEnvironment}]$
- $\text{Eval} < \text{Identifier} < t_\triangleright > x [x \text{ using } e] \equiv \text{Lookup}[x \text{ using } e]$
- $\text{Eval} < \text{Application} < t_1, t_2 > x [(\text{operator} \text{ operand}) \text{ using } e] \equiv$
  - $(\text{Eval} < \text{Expression} < t_1 \rightarrow t_2 > [\text{operator} \text{ using } e])$
  - // apply the value of operator to the value of operand
- $\text{Eval} < \text{Mapping} < t_1, t_2 > x [(x_1 \mapsto \text{body}) \text{ using } e] \equiv$
  - $x_2 : t_1 \mapsto \text{Eval} < \text{Expression} < t_2 > [\text{body} \text{ using } e \text{ bind}[x_1 \text{ to } x_2]]$
  - // eval body in a new environment with $x_1$ bound to $x_2$ as an extension of $e$
- $\text{Eval} < \text{FixExpression} < t_1, t_2 > x [\text{Fix} < t_1, t_2 > \text{ using } e] \equiv$
  - $g : (t_1 \rightarrow t_2) \mapsto \text{Fix} < t_1, t_2 > g$
  - // $\text{Fix} < t_1, t_2 > g$ is a fixed point of $g$

**Theorem.** $\forall[P \text{ predicateOn } \text{ Actor } t_1 \rightarrow t_2, F : \text{Immutable } t_1 \rightarrow t_2]$

$$\left( P[F[\bot < t_1, t_2 >]] \land \forall [g : t_1 \rightarrow t_2] P[g] \Rightarrow P[F[g]] \right) \Rightarrow P[\text{Fix} < t_1, t_2 > F]$$

// where $\bot < t_1, t_2 > \equiv x : t_1 \mapsto \bot < t_1, t_2 > x$

**Proof.** Follows immediately from Actor induction axioms applied to the behavior of $F$.

**Indeterminacy is foundational for digital computation**

**Sequential composition is not foundational because it can be defined as follows:**

- $\text{Eval} < \text{SequentialExpression} < t_\triangleright > [\{c ; x\} \text{ using } e] \equiv$
- $\text{Eval} < \text{Expression} < t_\triangleright > [x \text{ using Perform}[c \text{ using } e]]$
  - // where Perform[c using e] is the environment from performing c using e
On the other hand, non-determinacy can be defined using an implementation `JustOne` that arbitrates as follows:

**Chooser implements Choice**

```plaintext
Use[Scheduling],  // use Scheduling for this implementation

Contained[
  decided := TheNull Boolean;  // the variable decided is initialized as
  // the null of type Boolean

  pending ← New Suspended[1]]  // where out activity can be suspended

in 1 ↦ /* in1 message received */ {
  decided := Nullable True;  // decided is assigned to be a nullable of True
  Permit pending}  // permit an activity in pending to proceed (if there is one)

in 2 ↦ /* in2 message received */ {
  decided := Nullable False;  // decided is assigned to be a nullable of False
  Permit pending}  // permit an activity in pending to proceed (if there is one)

out ↦ /* out message received */ {
  // suspend in pending when IsNull decided;  // suspend in pending when IsNull[v]
  // Require: −IsNull[decided]
  UnNull v}  // return unNull of decided
```

Using Chooser, nondeterministic evaluation of the expression `(x1 either x2)` that either evaluates `x2` or evaluates `x2` can be defined as follows:

```plaintext
Eval<EitherExpression>(t ▷ •[x1 either x2] using e) ≡ {  
anArbiter ← New Chooser;  // identifier anArbiter is bound to a new Chooser
  anArbiter.in1,  // send in1 to anArbiter
  anArbiter.in2,  // while concurrently sending in2 to anArbiter
  anArbiter.out ?? // test the result when available
  True then Eval<Expression>(t ▷ •[x1 using e])  // if True then eval x1
  False then Eval<Expression>(t ▷ •[x2 using e])  // if False then eval x1
```

Consequently, Church/Turing nondeterministic execution can be defined using Actors, although the indeterminate execution of digital computation cannot in general be implemented using only nondeterministic execution, as shown in this article.

**Metatheory of the theory Actor**

MetaActor is a meta theory of Actor for proving theorems about Actor, which directly expresses provability of a proposition `Ψ` in the theory Actor using `Ψ ⊢_Actor`. (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 axioms.)

**Proof Checkers in the theory Actor**

A proof checker `pc:ProofChecker<|Actor|>` (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 Actor) means that `pc` is proof checker such that proposition `Ψ_1` infers proposition `Ψ_2` in the theory `Actor` (written `Ψ_1 ⊢_Actor pc Ψ_2`) such that:  

(\(Ψ_1 ⊢_Actor Ψ_2\)) ⇔ ∃[pc:ProofChecker<|Actor|>] Ψ_1 ⊢_Actor pc Ψ_2
Proof checking in the theory \texttt{Actor} is computationally decidable because:

\[ \forall [\text{Proposition } \Psi_1, \Psi_2; \text{pc:ProofChecker} \langle \text{Actor} \rangle] (\Psi_1 \vdash_{\text{Actor}} \text{pc} \Psi_2) \iff \text{pc} \Sigma [\Psi_1, \Psi_2] = \text{True} \]

where \( \Sigma [\Psi_1, \Psi_2] \) 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:

\[ \text{InductionChecker} \Sigma [\Psi_1, \Psi_2] = \Psi_1 \text{ ?? (} P[0] \land \forall [i : N] P[i] \Rightarrow P[+1[i]] \text{)} \text{ then } \Psi_2 = \forall [i : N] P[i], \text{ else False} \]

Note that InductionChecker correctly checks uncountably many instances of each of the theory \texttt{Actor} induction axioms.

The rule TheoremUse means that a theorem in \texttt{Actor} can be used in proofs in \texttt{Actor} as follows:

\[ \forall [\text{Proposition } \Psi_1, \Psi_2] \text{ TheoremUse} \Sigma [\Psi_1, \Psi_2] = \Psi_2 \text{ ?? (} \vdash_{\text{Actor}} \Psi_1 \text{) then } \text{True} \text{ else False} \]

Consequently, \( (\vdash_{\text{Actor}} \Psi) \vdash_{\text{Actor}} \text{TheoremUse} \Psi \). A consequence of TheoremUse is that unrestricted cut-elimination does not hold for the theory \texttt{Actor}.

There are uncountable proof checkers in the theory \texttt{Actor} 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\langle \text{t} > [c] \rangle where \text{t} is a type and c: \text{t} such that ForAllEliminationChecker\langle \text{t} > [c] \rangle \Sigma [\Psi_1, \Psi_2] = \Psi_1 \text{ ?? (} \forall [x : \text{t}] P[x] \text{)} \text{ then } \Psi_2 \text{=} P[c], \text{ else False}

Consequently, \( (\forall [x : \text{t}] P[x]) \vdash_{\text{Actor}} \text{ForAllEliminationChecker} \langle \text{t} > [c] \rangle P[c] \)

\textit{Types and propositions of the theory \texttt{Actor}}

Types and propositions of the theory \texttt{Actor} are axiomatized in terms of each other.

The following axioms hold for TypeIn\langle \text{Actor} \rangle (the type of types in the theory \texttt{Actor}) because types are intensional:

- \( N : \text{TypeIn} \langle \text{Actor} \rangle \) // \( N \) is type of natural numbers
- \( \forall [i : N] \text{ PropositionOfOrder} \langle \text{id} \rangle : \text{TypeIn} \langle \text{Actor} \rangle \)
- \( \forall [t_1, t_2, t_3, t_4 : \text{TypeIn} \langle \text{Actor} \rangle] \ (t_1, t_2) = (t_3, t_4) \Rightarrow t_1 = t_2 \land t_3 = t_4 \)
- \( \forall [t_1, t_2, t_3, t_4 : \text{TypeIn} \langle \text{Actor} \rangle] t_1 \rightarrow t_2 = t_3 \rightarrow t_4 \Rightarrow t_1 = t_2 \land t_3 = t_4 \) // computable procedures
- \( \forall [t_1, t_2, t_3, t_4 : \text{TypeIn} \langle \text{Actor} \rangle] t_2^t = t_4^t \Rightarrow t_1 = t_2 \land t_3 = t_4 \) // all functions
- \( \forall [t_1, t_2 : \text{TypeIn} \langle \text{Actor} \rangle; P_1, P_2, \text{predicateOn}_{\text{Actor}} t_1, P_1, \text{predicateOn}_{\text{Actor}} t_2, P_2] t_1 \equiv P_1 = t_2 \equiv P_2 \Rightarrow t_1 = t_2 \land P_1 = P_2 \)

For example, \( (N \rightarrow N) : \text{TypeIn} \langle \text{Actor} \rangle, \) etc.

The following induction axiom holds (cf. [Palmgren 1998, Uemura 2019]), which has uncountable instances:

\( \forall [P \text{ predicateOn}_{\text{Actor}} \text{ TypeIn} \langle \text{Actor} \rangle] \)

\( (P[\text{Actor}] \land \forall [i : N] P[\text{PropositionOfOrder}_{\text{Actor}} \langle \text{id} \rangle]) \)

\( \land \forall [t_1, t_2 : \text{TypeIn} \langle \text{Actor} \rangle] P[t_2] \Rightarrow P[t_1 \rightarrow t_2] \)

\( \land \forall [t_1, t_2 : \text{TypeIn} \langle \text{Actor} \rangle] P[t_2] \Rightarrow P[t_1 \rightarrow t_2] \)

\( \land \forall [t_1, t_2 : \text{TypeIn} \langle \text{Actor} \rangle] P[t_2] \Rightarrow P[t_2^t] \)

\( \land \forall [t : \text{TypeIn} \langle \text{Actor} \rangle] P[t] \Rightarrow P[t \text{ predicateOn}_{\text{Actor}} t] P[t \Delta P[t \vdash Q]] \)

\( \Rightarrow \forall [t : \text{TypeIn} \langle \text{Actor} \rangle] P[t] \)
Theorem Unique categoricity of \( \text{TypeIn} \langle \text{Actor} \rangle \), i.e., if \( \mathcal{M} \) is a type satisfying the theory \( \text{Actor} \), then there is a unique isomorphism \( I \) between \( \text{TypeIn} \langle \text{Actor} \rangle \) and \( \text{TypeIn}_\mathcal{M} \langle \text{Actor} \rangle \) is defined as follows:

- \( \forall [t_1, t_2] = [\llbracket t_1 \rrbracket, \llbracket t_2 \rrbracket]_\mathcal{M} \)
- \( \llbracket t_1 \rightarrow t_2 \rrbracket = \llbracket t_1 \rrbracket \rightarrow \llbracket t_2 \rrbracket \)
- \( \llbracket t_2 \uparrow t \rrbracket = \llbracket t_2 \rrbracket^{[t]} \)
- \( \llbracket t \exists P \rrbracket = \llbracket t \rrbracket \exists \llbracket P \rrbracket \)

The following induction axiom holds for propositions of the theory \( \text{Actor} \) (cf. [Martin-Löf 1998, Harrison 2017]), which has uncountable instances:

\[
(\forall [i: N, \ P \ \text{predicateOn}_{\text{Actor}} \ \text{PropositionOfOrder}_{\text{Actor}} \ \llbracket i \rrbracket])
\]

\[\forall [t: \text{TypeIn} \langle \text{Actor} \rangle; \ x_1, x_2: t] \ P\llbracket x_1=x_2 \rrbracket \wedge \forall [t_1, t_2: \text{TypeIn} \langle \text{Actor} \rangle; \ x: t] \ P\llbracket x: t \rrbracket \wedge \forall [\text{Proposition}_{\text{Actor}} \ \Psi_1, \ \Psi_2] \ P\llbracket \Psi_1 \wedge \Psi_2 \rrbracket \Rightarrow P\llbracket \Psi_1 \rrbracket \wedge P\llbracket \Psi_2 \rrbracket \wedge \forall [t: \text{TypeIn} \langle \text{Actor} \rangle, \ Q \ \text{predicateOn}_{\text{Actor}} \ t] \ (\forall [x: t] \ P\llbracket Q[x] \rrbracket) \Rightarrow P\llbracket \forall [x: t] \ Q[x] \rrbracket\]

\[\Rightarrow \forall [\text{Proposition}_{\text{Actor}} \ \Psi] \ P\llbracket \Psi \rrbracket\]

Theorem: Propositions of the theory \( \text{Actor} \) are characterized up to a unique isomorphism.

Inference in the theory \( \text{Actor} \)

Theorem: Deduction for the theory \( \text{Actor} \), i.e., the following holds:

\( \vdash_{\text{MetaActor}} \forall [\text{Proposition}_{\text{Actor}} \Phi, \Psi] (\vdash_{\text{Actor}} \Phi \Rightarrow \Psi) \iff (\Phi \vdash_{\text{Actor}} \Psi) \)

Proof. Suppose \( \vdash_{\text{Actor}} \Phi \Rightarrow \Psi \) and consequently \( \Phi \vdash_{\text{Actor}} \Psi \) by TheoremUse. Further suppose \( \Phi \) by ChainingForImplication and consequently \( \Phi \vdash_{\text{Actor}} \Psi \) by InferenceIntroduction.

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

Theorem: Inferential Adequacy, i.e., in MetaActor

\( \forall [\text{Proposition}_{\text{Actor}} \Psi] (\vdash_{\text{Actor}} \Psi) \Rightarrow \vdash_{\text{Actor}} \vdash_{\text{Actor}} \Psi \)

Proof. Suppose \( \vdash_{\text{Actor}} \Psi \). Let \( \vdash_{\text{Actor}} \Psi \) so that \( \text{pc1} \llbracket \Psi \rrbracket = \text{True} \). Then a provably total procedure \( \text{pc2:ProofChecker} \llbracket \text{Actor} \rrbracket \) can be defined such that \( \text{pc2} \llbracket \vdash_{\text{Actor}} \Psi \rrbracket = \text{True} \). Consequently,

\( \vdash_{\text{Actor}} \vdash_{\text{Actor}} \Psi \).

Theorem: If \( \mathcal{M} \) is a type satisfying the axioms of the theory \( \text{Actor} \), then there is a unique isomorphism \( \mathcal{M} \equiv_{\text{Model} \langle \text{Actor} \rangle} \).

Definition: \( \equiv \) can be defined by induction on propositions using the following where \( \mathcal{M} \) is used for \( \text{Actor} \) in \( \text{Actor}_\mathcal{M} \) (cf. [Tarski 1936]):

\( \forall [t: \text{TypeIn} \langle \text{Actor}_\mathcal{M} \rangle] \ (\equiv_\mathcal{M} \forall [x: t] \ P\llbracket x \rrbracket) \equiv \forall [x: t] \ \equiv_\mathcal{M} P\llbracket x \rrbracket \)

Theorem: The theory \( \text{Actor} \) is sound, i.e., \( \vdash_{\text{MetaActor}} (\vdash_{\text{Actor}} \Psi) \Rightarrow \vdash_{\text{Model} \langle \text{Actor} \rangle} \Psi \)

Proof. Follows from induction on types and propositions of \( \text{Actor} \).

Theorem: \( \text{Model} \llbracket \text{Actor} \rrbracket \) decides the theory \( \text{Actor} \), i.e.,

\( \vdash_{\text{MetaActor}} \forall [\Psi \ \text{proposition}_{\text{Actor}}] (\equiv_{\text{Model} \langle \text{Actor} \rangle} \Psi) \lor (\equiv_{\text{Model} \langle \text{Actor} \rangle} \neg \Psi) \)

Proof. Immediate from definition of \( \equiv_{\text{Model} \langle \text{Actor} \rangle} \),
Although Model \( \llbracket \text{Actor} \rrbracket \) decides the validity of each proposition of the theory \( \text{Actor} \), the theory is computationally and inferentially undecidable.

The predicate \( \text{Halt} \) can be defined as follows on deterministic Boolean expressions:

\[ \text{Halt}[x : \text{Deterministic } \llbracket \text{Boolean} \rrbracket] = \exists [y : \text{Boolean}] \ y = \text{Eval} x \]

Definitions.

- \( \mathcal{BExpression} \equiv \text{Deterministic } \llbracket \text{Expression} \rrbracket \llbracket \text{Boolean} \rrbracket \llbracket \text{String} \rrbracket \)
  // deterministic Boolean expression abstracted from a string

- \( \mathcal{BProcedure} \equiv \text{Deterministic } \llbracket \mathcal{BExpression} \rrbracket \llbracket \text{String} \rrbracket \rightarrow \text{Boolean} \llbracket \text{String} \rrbracket \)
  // deterministic Boolean procedure abstracted from a string on an expression abstracted from a string

- \( \text{Decider} \equiv \text{Total } \llbracket \mathcal{BProcedure} \rrbracket \)
  // total \( \mathcal{BProcedure} \)

Theorem. \( \text{Halt} \) is computationally undecidable on Boolean expressions abstracted from strings [Church 1935, Turing 1936], i.e., \( \forall [x : \mathcal{BExpression}] \ d\llbracket x \rrbracket = \text{True} \Leftrightarrow \text{Halt}[x] \)

Proof. Suppose to obtain a contradiction that \( d : \text{Decider} \) and \( \forall [x : \mathcal{BExpression}] \ d\llbracket x \rrbracket = \text{True} \Leftrightarrow \text{Halt}[x] \).

Let \( \text{SelfApplier} \equiv \text{Deterministic } \llbracket \text{SelfApplier} \rrbracket \rightarrow \text{Boolean} \llbracket \text{String} \rrbracket \) so that \( \text{SelfApplier} \) is a recursively defined type of a deterministic Boolean procedure abstracted from a string. Define the procedure \( \text{AntiDecider} : \text{SelfApplier} \) as follows where \( ( \) and \( ) \) are used to delimit an expression:

\( \text{AntiDecider} \equiv p : \text{SelfApplier} \rightarrow d\llbracket p(p) \rrbracket ?? \text{True then LoopForever} [], \text{False then True} \)

Consider the two possibilities for \( \text{Halt}[\llbracket \text{AntiDecider} \text{AntiDecider} \rrbracket] \) to obtain a contradiction as follows:

1. \( \text{Halt}[\llbracket \text{AntiDecider} \text{AntiDecider} \rrbracket] \)
   
  Thus \( d\llbracket \text{AntiDecider} \text{AntiDecider} \rrbracket = \text{True} \) and \( \neg \text{Halt}[\llbracket \text{AntiDecider} \text{AntiDecider} \rrbracket] \) by the definition of \( \text{AntiDecider} \), which is a contradiction.

2. \( \neg \text{Halt}[\llbracket \text{AntiDecider} \text{AntiDecider} \rrbracket] \)
   
   Thus \( d\llbracket \text{AntiDecider} \text{AntiDecider} \rrbracket = \text{False} \) and \( \text{Halt}[\llbracket \text{AntiDecider} \text{AntiDecider} \rrbracket] \) by the definition of \( \text{AntiDecider} \), which is a contradiction.

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

Theorem. Whether a proposition abstracted from a string is a theorem of \( \text{Actor} \) is computationally undecidable [Church 1935, Turing 1936], i.e., there does not exist a decider \( d \) for propositions of the theory \( \text{Actor} \) such that for every proposition \( \Psi \) of \( \text{Actor} \) abstracted from a string, \( d\llbracket \Psi \rrbracket = \text{True} \Leftrightarrow \vdash_{\text{Actor}} \Psi \)

Proof. Follows immediately from the computational undecidability of the halting problem for expressions abstracted from strings because of the following:

\( \forall [x : \mathcal{BExpression}] \ \text{Halt}[x] \Leftrightarrow \vdash_{\text{Actor}} \text{Halt}[x] \)

Theorem. The theory \( \text{Actor} \) is inferentially undecidable for propositions abstracted from strings, i.e., there is a proposition \( \Psi \) of \( \text{Actor} \) abstracted from a string such that \( (\not\vdash_{\text{Actor}} \Psi) \land (\not\vdash_{\text{Actor}} \neg \Psi) \).

Proof. Suppose to obtain a contradiction that the theory \( \text{Actor} \) is inferentially decidable for propositions abstracted from \( s \) and consequently \( \forall [x : \mathcal{BExpression}] \ (\vdash_{\text{Actor}} \text{Halt}[x]) \lor (\vdash_{\text{Actor}} \neg \text{Halt}[x]) \)

Only countably many instances of the induction axioms could have been used in the proofs because, the halting problem for expressions abstracted from strings is computationally decidable by computationally enumerating the proofs, which is a contradiction.

Theorem. There is a proposition \( \Psi \) of \( \text{Actor} \llbracket \text{String} \rrbracket \) such that \( \Psi \land \not\vdash_{\text{Actor}} \neg \Psi \).

Proof. By inferential undecidability let \( x : \mathcal{BExpression} \) be such that

\( (\not\vdash_{\text{Actor}} \text{Halt}[x]) \land (\not\vdash_{\text{Actor}} \neg \text{Halt}[x]) \). Therefore \( \neg \text{Halt}[x] \) because \( \text{Halt}[x] \Leftrightarrow \vdash_{\text{Actor}} \text{Halt}[x] \)

In practice, computational and inferential undecidability of provability, do not impose limitations on the ability to prove theorems for mathematical theories used in Intelligent Systems. Also, computational
and inferential undecidability of provability of the Actor theory of computation does not necessarily mean that the theory is “incomplete” in the sense that there are useful missing axioms because axioms of the theory characterize Model$\triangleright_{\text{Actor}}$ up to a unique isomorphism.

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

The theory Actor$\downarrow$String has the following induction axiom, which has countable instances because strings are countable:
\[
\forall[P\ \text{predicateOn}_{\text{Actor}}\downarrow\text{String} N] (P[0] \land \forall[j: N] P[j] \Rightarrow P[+1[j]]) \Leftrightarrow \forall[i: N] P[i]
\]

Definitions.
- Total$\triangleright_{\text{t}}$ = (Deterministic$\triangleright_{\text{N} \rightarrow \text{t}}$ $\exists$f $\forall[x: \text{N}] \exists[y: \text{t}] f[x] = y)$\downarrow$String
- ProvedTotal$\downarrow_{\text{Actor}}$String$\triangleright_{\text{t}}$ = Deterministic$\triangleright_{\text{N} \rightarrow \text{t}}$ $\exists$f $\forall[x: \text{N}] f[x] = y$
- Onto$\triangleright_{\text{t}}$ = Deterministic$\triangleright_{\text{N} \rightarrow \text{t}}$ $\exists$f $\forall[x: \text{N}] f[x] = y$
- ProvedEnumerator$\downarrow_{\text{Actor}}$String$\triangleright_{\text{t}}$ = ProvedTotal$\downarrow_{\text{Actor}}$String$\triangleright_{\text{t}}$ $\exists$f $\forall[x: \text{N}] f[x] = y$

Theorem. Theorem$\triangleright_{\text{Actor}}$String$\triangleright_{\text{t}}$ is computationally enumerable, i.e., there is a procedure Theorems$\downarrow_{\text{Actor}}$String$\triangleright_{\text{t}}$$\triangleright_{\text{t}}$.

Corollary. ProvedTotal$\downarrow_{\text{Actor}}$String$\triangleright_{\text{N}}$ is computationally enumerable, i.e., there is a procedure ProvedTotals$\downarrow_{\text{Actor}}$String$\triangleright_{\text{N}}$.

Definition. Define the procedure Diagonal$\triangleleft$Deterministic$\downarrow_{\text{N} \rightarrow \text{N}} \downarrow$String as follows:
\[
\text{Diagonal}[i: \text{N}] = 1 + (\text{ProvedTotals}[i][i])
\]

Lemma. Diagonal$\triangleright_{\text{Actor}}$String$\triangleright_{\text{N}}$

Proof. Suppose i: N. Let f: ProvedTotal$\downarrow_{\text{Actor}}$String$\triangleright_{\text{N}}$ = ProvedTotals$\triangleright_{\text{t}}$[i] and let $j: N = f[i]$. Therefore Diagonal$[i] = 1 + j$. Consequently, $\vdash_{\text{Actor}}$String Diagonal$\triangleright_{\text{N}}$.

Lemma. ¬Diagonal$\triangleright_{\text{Actor}}$String$\triangleright_{\text{N}}$

Proof. Diagonal differs from every ProvedTotal$\downarrow_{\text{Actor}}$String$\triangleright_{\text{N}}$ enumerated by ProvedTotals.

Theorem. The theory Actor$\downarrow$String is inconsistent [Church 1934], i.e.,
\[
\exists[\text{Proposition}_{\text{Actor}} \downarrow \text{String} \Psi] \neg \text{Actor} \downarrow \text{String} \Psi \land \neg \Psi
\]

Proof. Let $\Psi = \text{Diagonal} \triangleright_{\text{Actor}}$String ProvedTotal$\downarrow_{\text{Actor}}$String

The upshot is that the theory Actor is algorithmically inexhaustible, i.e., it is impossible to computationally enumerate theorems of the theory thereby reinforcing the intuition behind [Franzén, 2004]. According to [Church 1934], inconsistency of the theory Actor$\downarrow$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. 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 (Nondeterministic) Procedures, and Actors. Each theory $T$ has the following properties:

- $T$ is uniquely categorical for $Model_{T}$, i.e., if $X$ satisfies the axioms of $T$, then $X$ is isomorphic to $Model_{T}$, by a unique isomorphism.
- $T$ is sound, i.e., $\vdash T \Psi \iff \models Model_{T} \Psi$
- $Model_{T}$ decides each proposition of $T$, i.e., $\forall [Proposition_{T} \Psi] (\models Model_{T} \Psi) \lor (\not\models Model_{T} \Psi)$
- For all propositions $\Psi$ of $T$ and $p:ProofChecker_{T}$, $\vdash_{T} \Psi$ is computationally decidable.

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 categorial 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 uniquely categorical theories Act and O (the theory of the Ordinals) is based on the way that they are inductively constructed bottom-up.

• Monsters [Lakatos 1976] are unwanted elements in models of classical mathematical theories. 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 N) that can directly express proofs of theorems of classical mathematics including [Wiles 1995]. As shown, in this article, additional axioms are needed to axiomatize all digital computation up to a unique isomorphism. 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.

VI. 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 assumption has often held back the development of science.

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. 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]

Identifying and rectifying errors is fundamental to scientific progress. With respect to the subject matter of this article, 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 typically 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.

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 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 $\Psi$ is a proposition of theory $T$ if an 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. However, HOL Light 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.

The Church/Turing model is inadequate for digital computation, as explained in this article. Computing practice diverged from the Church/Turing model when external devices were attached to computers that interacted during computation, which is beyond the algorithmic Church/Turing model. [Hewitt, Bishop, and Steiger 1973] proposed Actors as the universal primitive for digital computation with soon-developed axioms [Greif 1975, Hewitt and Baker 1977, Yonezawa 1977, Hewitt and Atkinson 1979, Atkinson 1980]. [Clinger 1981] and [Agha 1986] developed denotational models but they did not characterize Actors up to a unique isomorphism, as in this article.

[Milner 1993] developed algebraic reduction for use in theories of computation. However even on the same chip, algebraic reduction is not possible for simultaneously sending and receiving a message to an Actor because in general for a message to be received it must go through arbitration with any other messages sent to the Actor. Synchronized message passing can be implemented as follows using a 2-phase commit protocol (cf. [Knabe 1992]) for an implementation I:

\[
\text{actor facet } I \text{ implements Synchronization} \langle I \rangle \rightarrow \text{ // synchronizer is a facet of I}
\]

\[
\text{actor function synchronize[aProvider]} \rightarrow \text{ // request received to synchronize with aProvider}
\]

\[
\text{(As I) } \langle \text{aProvider } \text{provide} \rangle \text{ // process message provided by aProvider in the}
\]

\[
\text{ // region of mutual exclusion of this Actor}
\]

In this way, the expression \( f \text{synchronize[AActor provide } \text{m} \rangle \) for the synchronizer facet \( f \) of an Actor \( x \) processes the message \( m \) synchronously. Synchronized messaging requires that the sender must wait to provide a message until it is requested and the recipient must wait for the message to be provided (meanwhile holding up processing of other messages). However, \( x \text{m} \) has neither of the extra wait times of synchronized communication nor the requirement that message passing must overlap in time for sender and receiver. Although algebra in the pi-calculus is elegant mathematics, synchronized message passing is not widely used in large software systems because it is slower and less robust than asynchronous message passing.

Also, algebraic refinement orderings [Hoare 2018] can be tricky for programming languages. For example, Interchange [Hoare 2018] does not hold in general for ActorScript [Hewitt and Woods assisted by Spurr 2019], e.g., behavior of the expression \( \{ y \leftarrow g_{*}[x], x \leftarrow f_{*}[4]; x+y \} \) (using "\", to mean concurrent execution so that \( g_{*}[x] \) and \( f_{*}[4] \) can be executed concurrently) does not contain the behavior of the expression \( \{ x \leftarrow f_{*}[4]; y \leftarrow g_{*}[x]; x+y \} \) with sequentially executed \( x \leftarrow f_{*}[4] \) and \( y \leftarrow g_{*}[x] \) interchanged from the former expression) because in the latter expression, \( x \) in \( g_{*}[x] \) comes from binding the identifier \( x \) in \( x \leftarrow f_{*}[4] \) instead of from an outer scope.

The Actor Model attempts to be as general as possible to support direct modeling and efficient implementation of all digital computation. For example, an Actor is not required to have an external mailbox as in Erlang [Armstrong, et. al. 1992]. Requiring an external mailbox is problematical for Actors because the mailbox would itself necessarily be another Actor thereby immediately leading to an infinite regress. Also, requiring the use of external mailboxes can slow message passing between Actors because it would always be necessary to first deposit a message in an Actor’s external mailbox so that the message could later be retrieved. Despite some inefficiency and lack of needed functionality (e.g., automatic reclamation of resources of unneeded processes), Erlang has been used to good effect in many impressive projects demonstrating the Actor paradigm.

Requiring use of external mailboxes or requiring use of a synchronous message passing could prevent achieving the goal of less than 10ns average send-to-receipt latency on a chip with thousands of general-purpose, high-performance cores for the next generation of Intelligent Systems [Hewitt 2019].
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 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 inferential 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.

ACKNOWLEDGMENTS

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. The uniquely categorical theories in this article build on the pioneering work of [Dedekind 1888].
APPENDIX: MATHMATICAl NOTATION

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

Mathematical *Proposition* is a discrimination of the following patterns:
- \( \neg \Psi_1, \Psi_1 \land \Psi_2: \text{PropositionOfOrder} \) where \( \Psi_1, \Psi_2: \text{PropositionOfOrder} \) and \( i: N_+ \)
- \( (x_1 = x_2): \text{PropositionOfOrder} \) where \( x_1, x_2: \text{Term} \) and \( t \) is a type
- \( (x:t): \text{PropositionOfOrder} \) where \( t \) is a type
- \( P^"x": \text{PropositionOfOrder} \) where \( x: \text{Term} \), \( t \) is a type and 
  \[ P: \text{Term} \rightarrow \text{Proposition} \] and \( i: N_+ \)
- \( (\Psi_1 + \Psi_2): \text{PropositionOfOrder} \) where \( i: N_+ \) and \( \Psi_1, \Psi_2: \text{PropositionOfOrder} \)
- \( (\Psi_1 \Rightarrow \Psi_2): \text{PropositionOfOrder} \) where \( p: \text{Term} \rightarrow \text{ProofChecker}, T: \text{Theory}, \Psi_1, \Psi_2: \text{PropositionOfOrder} \) and \( i: N_+ \)
- \( [s]: \text{PropositionOfOrder} \) is abstraction of \( s \) where \( s: \text{String} \rightarrow \text{PropositionOfOrder} \) with no free variables and \( i: N_+ \)
- \( [\Psi]: \text{String} \rightarrow \text{PropositionOfOrder} \) is quotation of \( \Psi \) where 
  \( \Psi: \text{PropositionOfOrder} \) and \( i: N_+ \).

Procedural *Expression* is a discrimination of the following:
- \( x: \text{Expression} \) where \( x: \text{Constant} \) and \( t \) is a type
- \( x: \text{Expression} \) where \( x: \text{Identifier} \) and \( t \) is a type
- \( [e_1, e_2]: \text{Expression} \) where \( e_1: \text{Expression} \), \( e_2: \text{Expression} \), and \( t_1 \) and \( t_2 \) are types
- \( (e_1 ? \text{True then} e_2, \text{False then} e_3): \text{Expression} \) where \( e_1: \text{Expression} \rightarrow \text{Boolean} \), \( e_2, e_3: \text{Expression} \) and \( t \) is a type
- \( (x:t_1 \rightarrow y): \text{Expression} \) where \( x: \text{Identifier} \), \( y: \text{Expression} \), and \( t_1 \) and \( t_2 \) are types
- \( x.m: \text{Expression} \) where \( m: \text{Expression} \), \( x \) is an Actor with a message handler with signature of type \( \text{Expression} \rightarrow \text{Term} \), and \( t_1 \) and \( t_2 \) are types
- \( [I[x_1, ..., x_n]]: \text{Expression} \) where \( I \) is an Actor implementation and \( x_1, ..., x_n \) are expressions.
- \( [s]: \text{Expression} \) is abstraction of \( s \) where \( s: \text{String} \rightarrow \text{Expression} \) with no free variables and \( t \) is a type
- \( [x]: \text{String} \rightarrow \text{Expression} \) is quotation of \( x \) where \( x: \text{Expression} \rightarrow \text{String} \), \( i: N_+ \) and \( t \) is a type.

Mathematical *Term* is a discrimination of the following patterns:
- \( x: \text{Term} \) where \( x: \text{Constant} \) and \( t \) is a type
- \( x: \text{Term} \) where \( x: \text{Variable} \) and \( t \) is a type
- \( [x_1, x_2]: \text{Term} \) where \( x_1: \text{Term} \), \( x_2: \text{Term} \), and \( t_1 \) and \( t_2 \) are types
- \( (x_1 ? \text{True then} x_2, \text{False then} x_3): \text{Term} \) where \( x_1: \text{Term} \rightarrow \text{Boolean} \), \( x_2, x_3: \text{Term} \) and \( t \) is a type \( (x: t_1 \rightarrow y): \text{Term} \) where \( x: \text{Variable} \), \( y: \text{Term} \), and \( t_1 \) and \( t_2 \) are types
- \( f[x]: \text{Term} \) where \( f: \text{Term} \rightarrow \text{Term} \), \( x: \text{Term} \), and \( t_1 \) and \( t_2 \) are types
The type of all sets restricted to ones that are not 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.

Berry’s Paradox can be formalized using the proposition Characterize\(\langle i \rangle [s, k] \) meaning that the string \( s \) characterizes the integer \( k \) as follows where \( i : \mathbb{N}_+ \):

- \( \text{Berry} \langle i \rangle \equiv (\text{Term} \langle \text{Proposition of Order} \rangle \langle i \rangle \langle N \rangle) \langle \text{String} \rangle \)
- Characterize\(\langle i \rangle [s : \text{Berry} \langle i \rangle], k : \mathbb{N} \] \( \equiv \forall [x : \mathbb{N}] [s]\langle x \rangle \leftrightarrow x = k \)

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:

\[
\text{BerryString} : \text{Berry} \langle i+1 \rangle \equiv \left[ \exists [s : \text{Proposition of Order} \langle i \rangle \langle \text{String} \rangle] \text{Length}[s] < 100 \Rightarrow \neg \text{Characterize} \langle i \rangle [s, j] \right]
\]

Note that

- \( \text{Length} [\text{BerryString}] < 100. \)
- \( \text{Berry} \langle i \rangle \vdash s \mapsto \text{Length}[s] < 100 \) is finite.
- Therefore, \( \text{BerryNumber} \) is finite where
  \( \text{BerryNumber} \equiv \exists [i : \mathbb{N}_+] \exists [s : \text{Berry} \langle i \rangle] \text{Length}[s] < 100 \land \text{Characterize} \langle i \rangle [s, j] \)
- \( \exists [i : \mathbb{N}_+] i \vdash \text{BerryNumber} \) because \( \mathbb{N}_+ \) is infinite.
- \( \exists \text{LeastBerry} \equiv \exists \text{Least} [\text{BerryNumber}] \)

However \( \text{BerryString} : \text{Berry} \langle i+1 \rangle \) **cannot be substituted** for \( s : \text{Berry} \langle i \rangle \). Consequently, the Berry Paradox as follows does not hold:

\( [\text{BerryString}] [\text{LeastBerry}] \leftrightarrow \neg \text{Characterize} \langle i \rangle [\text{BerryString}, \text{LeastBerry}] \)

**Wittgenstein** [Wittgenstein 1978]

Wittgenstein’s Paradox is blocked because the mapping \( \Psi \mapsto \psi \Psi \) does **not** have a fixed point (contra [Gödel 1931]) because the order of \( \psi \Psi \) 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 \( P \mapsto \left( \forall [i, j > i : N \not\Psi[j] \right) \) does not have a fixed point because the order of \( \forall [i, j > i : N \not\Psi[j] \right) \) is one great than the order of \( P \) since \( P \) is a predicate variable [cf. Priest 1997].

APPENDIX: ORDINALS AND NATURAL NUMBERS

Theorem 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 \( N \)
- \( \vdash 0 : N \) // +1 (successor) is of type \( N \)
- \( \forall [i : N] . \vdash i = 0 \Rightarrow i \not= 0 \) // 0 is not a successor
- \( \forall [i, j : N] . \vdash i = j \Rightarrow \vdash i = j \) // +1 is 1 to 1

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

\[ \forall [P \, \text{predicateOn} \, N] . (P[0] \land \forall [j : N] . P[j] \Rightarrow P[+1[j]]) \Rightarrow \forall [i : N] . P[i] \]

Theorem [cf. Dedekind 1888]: If \( M \) be a type satisfying the axioms of the theory \( N \), then there is a unique isomorphism \( I : M \models N \) defined as follows:

- Define by induction on \( \text{TypeIn} \langle N \rangle \)
  - \( I(0) \equiv 0 \)
  - \( I(t_1, t_2) \equiv \{ I(t_1), I(t_2) \} \)
  - \( I(t^2_1) \equiv I(t_1)^I \)
  - \( \forall [P \, \text{predicateOn} \, N] . \text{TypeIn} \langle N \rangle . I(t^P) \equiv I(t \supset P) \equiv I(t \supset I[P]) \)
- Define by induction on \( N \)
  - \( I(0) \equiv 0 \)
  - \( I(+[i]) \equiv +I([i]) \)
  - If \( x : [t_1, t_2], \) then \( I(x) \equiv [I(1^{st}[x]), I(2^{nd}[x])] \)
  - If \( x : t^2, \) then \( I(x) = y : I(t_2) \mapsto I(x[I^1[y]]) \) // inductive hypothesis for \( I \) on \( t_2 \)

I is a unique isomorphism because of the following:
- \( I \) is defined on \( \text{TypeIn} \langle N \rangle \)
- \( I \) is 1-1
- \( I \) is onto \( M \)
- \( I \) is a homomorphism
- \( I^1 \) is a homomorphism
- If \( g \) is an isomorphism of \( Model \langle N \rangle \) with \( M \), then \( g = I \)
Corollary There are no infinite numbers (monsters) in models of the theory \( \mathbb{N} \), e.g., if \( M \) satisfies the axioms of the theory \( \mathbb{N} \) for \( N \), then \( \exists [j: M] \forall [i: N] i < j \).

**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 \( \mathbb{N} \) (where \( \forall [\alpha, \beta: \mathcal{O}] \alpha \beta \leftrightarrow \alpha < \beta \)):

- \( \forall [\alpha: \mathcal{O}] \alpha < 0 \) \hspace{1cm} // 0 is an ordinal
- \( \forall [\alpha: \mathcal{O}] \alpha < 0 \) \hspace{1cm} // 0 has no predecessor
- \( +1: \mathcal{O} \to \mathcal{O} \) \hspace{1cm} // \( +1[\alpha] = \alpha + 1 \)
- \( \forall [\alpha, \beta: \mathcal{O}] \alpha < \beta \land \beta < \gamma \Rightarrow \alpha < \gamma \) \hspace{1cm} // \( < \) is transitive
- \( \forall [\alpha, \beta, \mathcal{O}] \alpha < \beta \lor \alpha = \beta \lor \alpha > \beta \) \hspace{1cm} // trichotomy
- \( \forall [P \text{ predicateOn}_{\mathcal{O}} \mathcal{O}] \) \( \text{Least}[P]: \mathcal{O} \) axiomatized by cases as follows:
  1. \( \exists [\alpha: \mathcal{O}] (P[0] \land \forall [\alpha: \mathcal{O}] P[\alpha] \Rightarrow \text{Least}[P] \leq \alpha) \)
  2. \( \exists [P \text{ predicateOn}_{\mathcal{O}} \mathcal{O}] \) \( \forall [\alpha: \mathcal{O}] \forall [\beta: \mathcal{O}] (P[\beta] \Rightarrow P[\alpha]) \Rightarrow \forall [\alpha: \mathcal{O}] P[\alpha] \)
- \( \forall [\alpha: \mathcal{O}] \omega_{\alpha}: \mathcal{O} \) axiomatized by ordinal induction as follows:
  3. \( \omega_{0} = 0 \)
  4. \( \omega_{\alpha + 1} = \text{Least}[\beta \mapsto 1 \text{To} 1[\beta \rightarrow \omega_{\alpha + 1} \land 1 \text{To} 1[\omega_{\alpha + 1}, \mathbb{N}]]] \)
  5. \( \omega_{\alpha} = \text{Least}[\beta \mapsto \forall [\gamma: \mathcal{O}] \beta < \gamma \land \forall [\gamma: \mathcal{O}] \gamma < \omega_{\alpha}] \)

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

\( \forall [P \text{ predicateOn}_{\mathcal{O}} \mathcal{O}] (P[0] \land \forall [\alpha: \mathcal{O}] \forall [\beta: \mathcal{O}] P[\beta] \Rightarrow P[\alpha]) \Rightarrow \forall [\alpha: \mathcal{O}] P[\alpha] \)

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

**Theorem:** If \( M \) is a type satisfying the axioms of the theory \( \mathcal{O} \), then there is a unique isomorphism \( I: \mathcal{M}^\text{Model}_<\mathcal{O}> \) defined as follows:

- Define by induction on \( \text{TypeIn}_<\mathcal{O}> \)
  - \( I[\mathcal{O}] = M \)
  - \( I[\! t_1, t_2 \!] = [I[t_1], I[t_2]]_M \)
  - \( I[t_2^t] = I[t_2]^{[t^t]} \)
  - \( \forall [P \text{ predicateOn}_{\mathcal{O}} \text{TypeIn}_<\mathcal{O}>] I[t \exists P] = I[t] \exists I[P] \)
- Define by induction on \( \text{TypeIn}_<\mathcal{O}> \)
  - Define by ordinal induction on \( \mathcal{O} \)
    - \( I[0] = 0_M \)
    - \( I[\alpha + 1] = I[\alpha] + M \)
    - \( I[\alpha] = \text{Least}_M[\beta \mapsto \beta \geq M[\alpha]] \)
  - if \( x: [t_1, t_2] \), then \( I[x] = [I[1^x[x]], I[2^x[x]]]_M \)
  - if \( x: t_2^t \), then \( I[x] = y: t_2 \rightarrow I[x[1^y[y]]] \) \hspace{1cm} // inductive hypothesis for \( 1 \) on \( t_2 \)

\( \text{TypeIn}_<\mathcal{O} > \) is a strict generalization of sets in \( 1^\text{st}-\text{order} \) set theory, e.g., \( \mathbb{N} \) is \textbf{not} in the cumulative hierarchy of sets.
REFERENCES


H. Geuvers *(In)consistency of Extensions of Higher Order Logic and Type Theory* Types. April 18-21, 2006.


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


F. Knabe *A Distributed Protocol for Channel-Based Communication with Choice* PARLE’92.


A. Tarski. *Der Wahrheitsbegriff in den formalisierten Sprachen* Studia Philosophica. 1. 1936