What is computation?
Actor Model versus Turing's Model

Carl Hewitt
http://carlhewitt.info

Concurrency is of crucial importance to the science and engineering of computation in part because of the rise of the Internet and many-core architectures. However, concurrency extends computation beyond the conceptual framework of Church, Gandy [1980], Gödel, Herbrand, Kleene [1987], Post, Rosser, Sieg [2008], Turing, etc. because there are effective computations that cannot be performed by Turing Machines.

In the Actor model [Hewitt, Bishop and Steiger 1973; Hewitt 2010], computation is conceived as distributed in space where computational devices communicate asynchronously and the entire computation is not in any well-defined state. (An Actor can have stable information about what it was like when a message was received.) Turing's Model is a special case of the Actor Model.

A non-deterministic Turing Machine has bounded non-determinism (i.e. there is a bound on the size of integer that can be computed starting on a blank tape by an always-halting machine). Proving that a server will actually provide service to its clients requires unbounded non-determinism. In the semantics of bounded nondeterminism, a request to a shared resource might never receive service because a nondeterministic transition is always made to service another request instead. That's why the semantics of CSP were reversed from bounded non-determinism [Hoare CSP 1978] to unbounded non-determinism [Hoare CSP 1985]. However, bounded non-determinism was but a symptom of deeper underlying issues with communicating sequential processes as a foundation for concurrency. The Computational Representation Theorem [Clinger 1981, Hewitt 2006] characterizes the semantics of Actor Systems without making use of sequential processes.

In his Turing lecture, Robin Milner wrote: "*Now, the pure lambda-calculus is built with just two kinds of thing: terms and variables. Can we achieve the same economy for a process calculus? Carl Hewitt, with his Actors model, responded to this challenge long ago; he declared that a value, an operator on values, and a process should all be the same kind of thing: an Actor. This goal impressed me, because it implies the homogeneity and completeness of expression ... So, in the spirit of Hewitt, our first step is to demand that all things denoted by terms or accessed by names--values, registers, operators, processes, objects--are all of the same kind of thing...*"

However, there remains a Great Divide between process calculi and the Actor Model:

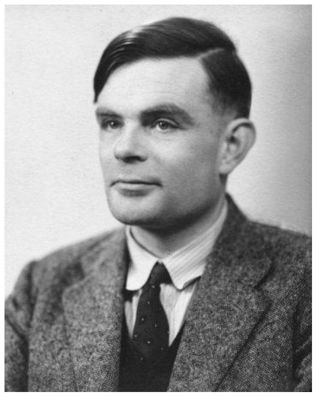
- Process calculi: communication using channels, algebraic equivalence, bi-simulation [Park 1980], etc.
- Actor Model: communication using Actor addresses, futures [Baker and Hewitt 1977], Swiss cheese [Hewitt and Atkinson 1979], garbage collection, etc.

A historical background discusses roles played by Alonzo Church, Ole-Johan Dahl, Edsger Dijkstra, Frederick Fitch, Kurt Gödel, Tony Hoare, Robin Milner, Kristen Nygaard, Dana Scott, Alan Turing, Ludwig Wittgenstein, and others in the development of these ideas.

A Computable Universe: Understanding Computation & Exploring Nature as Computation. Dedicated to the memory of Alan M. Turing on the 100th anniversary of his birth. Edited by Hector Zenil. World Scientific Publishing Company. 2012

Contents

Turing's Model


Alan Turing

Turing’s [1936] model of computation was intensely individual and sequential in that:

- *"the behavior of the computer at any moment is determined by the symbols which he* [the computer] *is observing, and his ‘state of mind’ at that moment"*
- *"there is a bound B to the number of symbols or squares which the computer can observe at one moment. If he wishes to observe more, he must use successive observations*."

In the above, computation was conceived as being carried out in a single place by a device that proceeds from one well-defined state to the next while carrying out a calculation.[i]

Turing [1948] stated the following thesis:

> *LCMs* [logical computing machines: Turing's expression for Turing machines] *can do anything that could be described as … "purely mechanical"…This is sufficiently well established that it is now agreed amongst logicians that "calculable by means of an LCM" is the correct accurate rendering* [of phrases like "purely mechanical"]

Kurt Gödel

Kurt Gödel declared that

> It is “*absolutely impossible that anybody who understands the question* [What is computation?] *and knows Turing’s definition should decide for a different concept.”*

Actor Model

However, Turing's model is in need of revision because of the increasing importance of concurrency in systems implemented using client-cloud computing and many-core computer architectures.

In the Actor model [Hewitt, Bishop and Steiger 1973; Hewitt 2010], computation is conceived as distributed in space where computational devices called Actors communicate asynchronously using addresses of Actors and the entire computation is not in any well-defined state. (An Actor can have information about other Actors that it has received in a message about what it was like when the message was sent.) The behavior of an Actor is defined when it receives a message and at other times may be indeterminate.

Axioms of locality including *Structural* and *Operational* hold as follows [Baker and Hewitt 1977]:[ii]

- *Structural:* The local storage of an Actor can include *addresses* only
    1. that were provided when it was created
    2. that have been received in messages
    3. that are for actors created here
- *Operational:* In response to a message received, an Actor can
    1. create more Actors
    2. send messages to addresses in the following:
        - the message it has just received
        - its local storage
    3. Specify how to process another message

The Actor Model differs from its predecessors and most current models of computation in that the Actor model assumes the following:

- Concurrent execution in processing a message.
- The following are *not* required by an Actor: a thread, a mailbox, a message queue, its own operating system process, *etc.*
- Message passing has the same overhead as looping and procedure calling.

Alonzo Church

Functions defined by lambda expressions [Church 1941] are special case Actors that never change. That Actors which behave like mathematical functions exactly correspond with those definable in the lambda calculus provides an intuitive justification for the rules of the lambda calculus:[1]

- *Lambda identifiers*: each identifier is bound to the address of an Actor. The rules for free and bound identifiers correspond to the Actor rules for addresses.
- *Beta reduction*: each beta reduction corresponds to an Actor receiving a message. Instead of performing substitution, an Actor receives addresses of its arguments.

The lambda calculus can express parallelism but not general concurrency (see discussion below).

Turing's Model is a special case of the Actor Model

Actor systems can implement systems that are impossible in Turing's model as illustrated by the following standard example of "*unbounded nondeterminism*":[iii]

There is a bound on the size of integer that can be computed by an *always-halting* nondeterministic Turing Machine starting on a blank tape.

Gordon Plotkin [1976] gave an informal proof as follows:[iv]

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.*

Consequently, *either*

- The tree has an infinite path. ⇔ The tree is infinite. ⇔ It is possible that P does not halt.
  If it is possible that P does not halt, then it is possible that that the set of outputs with which P halts is infinite.

or

- The tree does not have an infinite path. ⇔ The tree is finite. ⇔ P always halts.
  If P always halts, then the tree is finite and the set of outputs with which P halts is finite.

1 Note that in the definition in ActorScript [Hewitt 2011] of the lambda-calculus below:

- All operations are local.
- The definition is modular in that each lambda calculus programming language construct is an Actor.
- The definition is easily extensible since it is easy to add additional programming language constructs.
- The definition is easily operationalized into efficient concurrent implementations.
- The definition easily fits into more general concurrent computational frameworks for many-core and distributed computation.

```
identifier(x) ≡
        eval (environment) → environment.lookup (x)
        bind (binding, environment) → lookup (y) → y ?? (x → binding, else →environment.lookup (y))

application(operator, operand) ≡
   eval (environment) →  operator.eval (environment)(operand.eval (environment))

lambda(formal, body) ≡
   eval (environment) →  (argument) →  body.eval (formal.bind ( argument, environment))
```

By contrast, there are always-halting Actor systems with no inputs that can compute an integer of unbounded size:[2]

An Actor is created with local storage that is initialized with an integer variable *count* initialized to **0** and a Boolean variable *continue* that is **true** with the following behavior:

- When a ***stop*** message is received, set *continue* to **false** and return *count*.
- When a ***go*** message is received:
  1. if *continue* is **true**, increment *count* by **1** and send myself a ***go*** message.
  2. if *continue* is **false**, do nothing

The above Actor is started by concurrently sending it both a ***go*** message and a ***stop*** message.

The above Actor system can be implemented using ActorScript™ [Hewitt 2010a] as follows:

```
Unbounded ≡[3]
 start  →                  ⓘ a start message is implemented by
      let c ← Counter(0) →
         ⓘ let c be a Counter that is a created by Counter with count equal  0 and continue equal true
        {c.go ,           ⓘ send c a go message and concurrently
         c.stop }    ⓘ return the value of sending c a stop message

Counter ≡
 actor(count ↦[4]Integer) continue = true  (|[5]
  stop  → count  also continue = false,[6]        ⓘ return count also continue becomes false
  go  →                                         ⓘ a go message does
   continue  ??[7] (
       true →            ⓘ if continue is true then
           exit[8]  go  also count = count+1,[9]  ⓘ exit sending self a go message also count is incremented
       false → void)[10]|)[11]                   ⓘ if continue is false return void
```

By the semantics of the Actor model of computation [Clinger 1981, Hewitt 2006] sending Unbounded a *start* message results in sending an integer of unbounded size to the return address received with the *start* message**.**

[2] Plotkin's proof does not apply to the Actor system below for the following reason: In order to produce an output, the Actor System must pass through a sequence of interactions with ***go*** messages that it receives. However, during these interactions with ***go*** messages, the system is not in a well-defined state because there is a ***stop*** message in transit (perhaps in the physical form of photons). Consequently, the computation is inherently concurrent and not the sequence of global states assumed in Plotkin's proof for a Nondeterministic Turing Machine.

[3] read as "*is defined to be*"

[4] read as "*has type*"

[5] token that marks beginning of Actor's methods (i.e. message handlers)

[6] token that separates methods

[7] read as "*has cases*"

[8] exit and return following value

[9] token that separates condition handlers

[10] read as "end cases"

[11] token that marks end of Actor's methods

Nondeterminism is a special case of Indeterminism

Consider the following Nondeterministic Turing Machine that starts at *Step 1*:

```
Step 1: Either print 1 on the next square of tape or execute Step 3.
Step 2: Execute Step 1.
Step 3: Halt
```

According to the definition of Nondeterministic Turing Machines, the above machine might never halt.[v]

Note that the computation performed by the above machine are structurally different from the Counter program that implements unbounded nondeterminism in the following way:

1. The decision making of the above Nondeterministic Turing Machine is *internal* (having an essentially individual psychological basis).
2. The decision making of the above Actor counter is partly external (having an essentially sociological and anthropological basis)

Unbounded nondeterminism may seem like an esoteric property, but it is crucial to showing that a server does not accidentally starve a client by always serving others instead. And there are many other systems (e.g. computer operating systems) that cannot be implemented using Turing machines.

Actors are becoming the default model of computation. C#, Java, JavaScript, and Objective C are all headed in the direction of the Actor Model and ActorScript is a natural extension of these languages. Since it is very close to practice, many programmers just naturally assume the Actor Model.

The following major developments in computer technology are pushing the Actor Model forward because Actor Systems are highly scalable:

- Many-core computer architectures
- Client-cloud computing

In fact, the Actor Model and ActorScript can be seen as codifying what are becoming some best programming practices for many-core and client-cloud computing.

Configurations versus Global States

Computations are represented differently in State Machines and Actors:

1. *State Machine*: a computation can be represented as a global state that determines all information about the computation. It can be nondeterministic as to which will be the next global state, *e.g.*, in simulations where the global state can transition nondeterministically to the next state as a global clock advances in time, e.g., Simula [Dahl and Nygaard 1967].[vi]
2. *Actors*: a computation can be represented as a configuration. Information about a configuration can be indeterminate.[12]

Kristen Nygaard (left)
Ole Johan Dahl (right)

In 1975, Irene Greif published the first operational model of Actors in her dissertation. Two years after Greif published her operational model, Carl Hewitt and Henry Baker published the Laws for Actors [Baker and Hewitt 1977].

[12] For example, there can be messages in transit that will be delivered at some indefinite time.

The *Computational Representation Theorem* [Clinger 1981; Hewitt 2006] characterizes computation for systems which are closed in the sense that they do not receive communications from outside:

The denotation $\mathsf{Denote}_{\mathbf{S}}$ of a closed system $\mathbb{S}$ represents all the possible behaviors of $\mathbb{S}$ as[vii]

$$\mathsf{Denote}_{\mathsf{S}} = \mathrm{limit}_{i\to\infty} \mathrm{Progression}_{S}{}^{i}$$

where $\mathrm{Progression}_{\mathrm{S}}$ takes a set of partial behaviors to their next stage, i.e., $\mathrm{Progression}_{\mathbf{S}}{}^{\mathbf{i}} \rightarrow$[13] $\mathrm{Progression}_{\mathbf{S}}{}^{\mathbf{i+1}}$

In this way, $\mathbb{S}$ can be mathematically characterized in terms of all its possible behaviors (including those involving unbounded nondeterminism).[14]

The denotations form the basis of constructively checking programs against all their possible executions,[15]

A consequence of the Computational Representation system is that an Actor can have an *uncountable* number of different possible outputs. For example, Real.***go*** can output any real number[16] between 0 and 1 where

Real ≡ ***go*** → **[**(0 **either** 1)**]** ## **postpone** Real.***go***

where

- (0 **either** 1) is the nondeterministic choice of 0 or 1,
- **[**first**]** ## rest is the sequence that begins with first and whose remainder is rest, and
- **postpone** expression delays execution of expression until the value is needed.

The upshot is that *concurrent systems can be represented and characterized by logical deduction but cannot be implemented.* Thus, the following practical problem arose:

How can practical programming languages be rigorously defined since the proposal by Scott and Strachey [1971] to define them in terms λ-calculus failed because the λ-calculus cannot implement concurrency?

A proposed answer to this question is the semantics of ActorScript [Hewitt 2010].

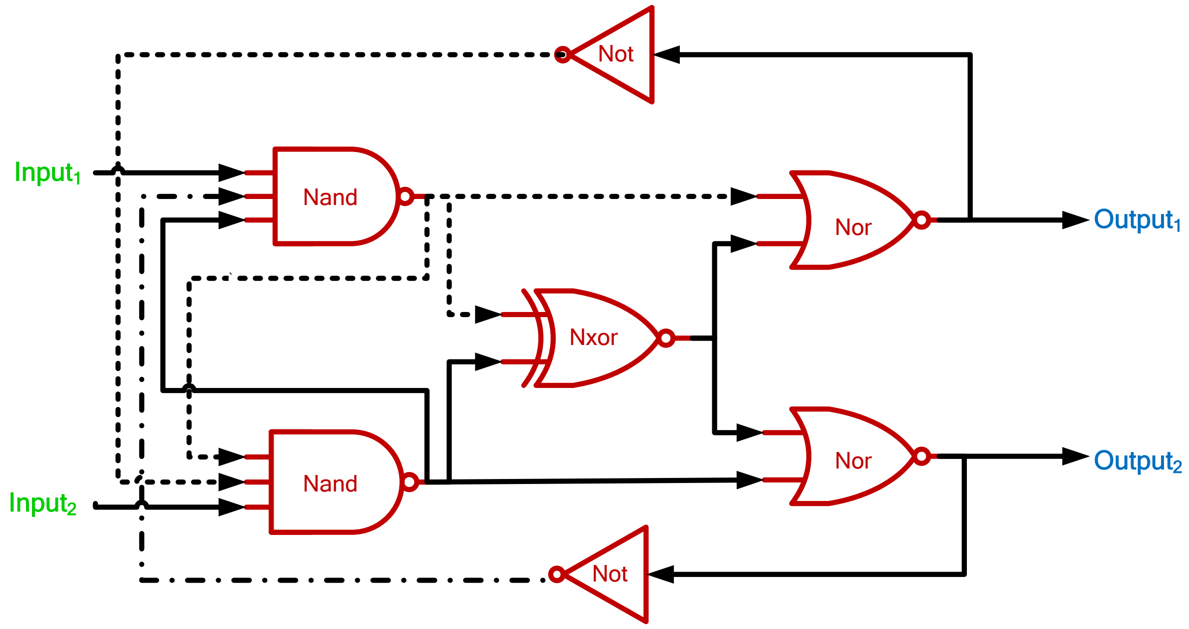
13 read as "*can evolve to*"

14 There are no messages in transit in $\mathsf{Denote}_{\mathsf{S}}$

15 a restricted form of this can be done via Model Checking in which the properties checked are limited to those that can be expressed in Linear-time Temporal Logic [Clarke, Emerson, Sifakis, *etc*. ACM 2007 Turing Award]

16 using binary representation. See [Feferman 2012] for more on computation over the reals.

Another point of departure from Turing's model is that concurrency violates a narrowly conceived "public processes" [Hofstadter 1980] criterion for computation. Actor systems make use of hardware devices called arbiters to decide the order of processing messages.



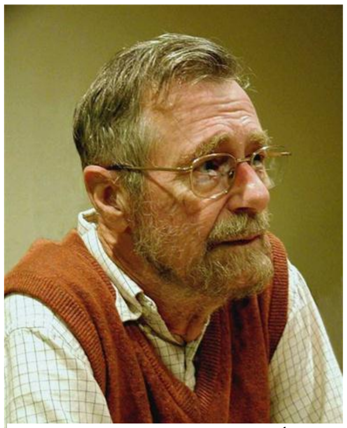
After the above circuit is started, it can remain in a meta-stable state for an unbounded period of time before it finally asserts either Output1$_1$ or Output$_2$. Examples of the operation of the above circuit are given in the endnotes.[viii]

The output of the operation of an Arbiter can in general *not* be logically inferred from its inputs. Thus (contrary to the claim of [Kowalski 1988]), computation is *not* subsumed by deduction. Consequently, Logic Programming[17] (although sometimes useful) is not a universal programming paradigm. See the appendices of this paper for an overview of modern inconsistency-robust Logic Programming.

The internal processes of arbiters are not public processes. Attempting to observe them affects their outcomes. Instead of observing the internals of arbitration processes, we necessarily await outcomes. Indeterminacy in arbiters produces indeterminacy in Actors. The reason that we await outcomes is that we have no realistic alternative.[ix]

[17] See [Hewitt 2009-2011] for the middle history of Logic Programming.

Unbounded Nondeterminism Controversy


Edsger Dijkstra[1]

Considerable controversy developed over issues involving unbounded nondeterminism.

Dijkstra believed that unbounded nondeterminism is impossible to implement

Edsger Dijkstra believed that unbounded nondeterminism cannot be implemented.[x] His belief was manifested in his theory of computation based on "*weakest preconditions*" for global states of computation [Dijkstra 1976] in which he argued that unbounded nondeterminism results in non-continuity of his weakest precondition semantics.[17]

Bounded nondeterminism in CSP

Hoare was convinced that unbounded nondeterminism could not be implemented and so the semantics of CSP specified bounded nondeterminism.

Consider the following program written in CSP [Hoare 1978]:

```
[X :: Z!stop( )    ⓘ In process X, send Z a stop message
  ||   ⓘ  process X operates in parallel with process Y
 Y :: guard: boolean; guard := true;   ⓘ In process Y, initialize boolean variable guard to true and then
       *[guard→ Z!go( ); Z?guard]  ⓘ while guard is true, send Z a go message and then input guard from Z
  ||        ⓘ  process Y operates in parallel with process Z
 Z :: n: integer; n:= 0; ⓘ In process Z, initialize integer variable n to 0 and then
           continue: boolean; continue := true;      ⓘ initialize boolean variable continue to true and then
       *[              ⓘ repeatedly either
           X?stop( ) → continue := false; ⓘ  input a stop message from X, set continue to false and then
                        Y!continue;          ⓘ  send Y the value of continue
              []                             ⓘ or
          Y?go( )→ n := n+1;                 ⓘ  input a go message from Y, increment n, and then
                   Y!continue]]              ⓘ send Y the value of continue
```

According to Clinger [1981]:

> *this program illustrates global nondeterminism, since the nondeterminism arises from incomplete specification of the timing of signals between the three processes X, Y, and Z. The repetitive guarded command in the definition of Z has two alternatives: either the stop message is accepted from X, in which case continue is set to false, or a go message is accepted from Y, in which case n is incremented and Y is sent the value of continue. If Z ever accepts the stop message from X, then X terminates. Accepting the stop causes continue to be set to false, so after Y sends its next go message, Y will receive false as the value of its guard and will terminate. When both X and Y have terminated, Z terminates because it no longer has live processes providing input.*
>
> *As the author of CSP points out, therefore, if the repetitive guarded command in the definition of Z were required to be fair, this program would have unbounded nondeterminism: it would be guaranteed to halt but there would be no bound on the final value of n. In actual fact, the repetitive guarded commands of CSP are not required to be fair, and so the program may not halt [Hoare 1978][18]. This fact may be confirmed by a tedious calculation using the semantics of CSP [Francez, Hoare, Lehmann, and de Roever 1979] or simply by noting that the semantics of CSP is based upon a conventional power domain and thus does not give rise to unbounded nondeterminism.*

[18] A very important point is that Actors do not have to make use of the repeated nondeterministic choices of CSP as in loop above: *[X?stop( ) → continue := false; Y!continue; [] Y?go( ) → n := n+1; Y!continue]. The nondeterministic choice of sources of input poses fundamental practical and theoretical problems [Knabe 1992].

The upshot was that Hoare was convinced that unbounded nondeterminism is impossible to implement. That's why the semantics of CSP specified bounded nondeterminism. But Hoare knew that trouble was brewing in part because for several years proponents of the Actor Model had been beating the drum for unbounded nondeterminism. To address this problem, he suggested that implementations of CSP should be as close as possible to unbounded nondeterminism!

However, using the above semantics for CSP it was impossible to formally prove that a server actually provides service to multiple clients[19] (as had been done previously in the Actor Model). That's why the semantics of CSP were reversed from bounded non-determinism [Hoare CSP 1978] to unbounded non-determinism [Hoare CSP 1985].[xi] Bounded non-determinism was but a symptom of deeper underlying issues with nondeterministic transitions in communicating sequential processes (see [Knabe 1992]).

Summary of the Unbounded Nondeterminism Controversy

A nondeterministic system is defined to have "*unbounded nondeterminism*" exactly when both of the following hold:

1. When started, the system always halts.
2. For every integer n, it is possible for the system to halt with output that is greater than n.

This article has discussed the following points about unbounded nondeterminism controversy:

- A Nondeterministic Turing Machine cannot implement unbounded nondeterminism.[xii]
- Dijkstra believed that unbounded nondeterminism cannot be implemented.
- Semantics of unbounded nondeterminism are required to prove that a server provides service to every client.
- An Actor system [Hewitt, *et. al.* 1973] can implement servers that provide service to every client and consequently unbounded nondeterminism.
- The semantics of CSP [Francez, Hoare, Lehmann, and de Roever 1979] specified bounded nondeterminism for reasons mentioned in the article. Since Hoare *et. al.* wanted to be able to prove that a server provided service to clients, the semantics of a subsequent version of CSP were switched from bounded to unbounded nondeterminism.
- Unbounded nondeterminism was but a symptom of deeper underlying issues with communicating sequential processes as a foundation for concurrency.[xiii]

[19] In the semantics of bounded nondeterminism, a request to a shared resource might never receive service because a nondeterministic choice is always made to service another request instead.

Process Calculi

In his Turing lecture, Robin Milner [1993] wrote:[xiv]

> *Now, the pure lambda-calculus is built with just two kinds of thing: terms and variables. Can we achieve the same economy for a process calculus? Carl Hewitt, with his Actors model, responded to this challenge long ago; he declared that a value, an operator on values, and a process should all be the same kind of thing: an Actor.*
>
> *This goal impressed me, because it implies the homogeneity and completeness of expression ...*
>
> *So, in the spirit of Hewitt, our first step is to demand that all things denoted by terms or accessed by names--values, registers, operators, processes, objects--are all of the same kind of thing….*

Robin Milner

Process calculi (*e.g.* [Milner 1993; Cardelli and Gordon 1998]) are closely related to the Actor model. There are similarities between the two approaches, but also many important differences (philosophical, mathematical and engineering):

- There is only one Actor model (although it has numerous formal systems for design, analysis, verification, modeling, etc.) in contrast with a variety of species of process calculi.
- The Actor model was inspired by the laws of physics and depends on them for its fundamental axioms in contrast with the process calculi being inspired by algebra [Milner 1993].
- Unlike the Actor model, the sender is an intrinsic component of process calculi because they are defined in terms of reductions (as in the λ-calculus).
- Processes in the process calculi communicate by sending messages either through named channels (synchronous or asynchronous), or via ambients (which can also be used to model channel-like communications [Cardelli and Gordon 1998]). In contrast, Actors communicate by sending messages to the addresses of other Actors (this style of communication can also be used to model channel-like communications using a two-phase commit protocol [Knabe 1992]).

There remains a Great Divide between process calculi and the Actor Model:

- *Process calculi:* communication using channels, algebraic equivalence, bi-simulation [Park 1980], *etc*.
- *Actor Model:* communication using Actor addresses, futures [Baker and Hewitt 1977], Swiss cheese [Hewitt and Atkinson 1979], garbage collection, *etc.*

Computational Undecidability

Some questions cannot be uniformly answered computationally.

Halting Problem

The halting problem is to computationally decide whether a program halts on a given input[20] *i.e.,* there is a total computational deterministic predicate Halt such that the following 3 properties hold for any program p and input x:

1. $\text{Halt}(p, x) \rightarrowtail_1 \textit{True} \;\Leftrightarrow\; \underline{\downarrow}(\lfloor p \rfloor(x))$
2. $\text{Halt}(p, x) \rightarrowtail_1 \textit{False} \;\Leftrightarrow\; \neg\underline{\downarrow}(\lfloor p \rfloor(x))$
3. $\text{Halt}(p, x) \rightarrowtail_1 \textit{True} \;\vee\; \text{Halt}(p, x) \rightarrowtail_1 \textit{False}$

[Church 1936 and later Turing 1936] published proofs that the halting problem is computationally undecidable.[xv]

Theorem: $\vdash \neg$ComputationallyDecidable[HaltingProblem][21]

The proof method of showing computationally undecidability developed by Turing and Church demonstrates limits of inference including *Inferential Undecidability* .

[20] Adapted from [Church 1936]. Normal forms were discovered for the lambda calculus, which is the way that they "halt." [Church 1936] proved the halting problem computationally undecidable. Having done considerable work, Turing was disappointed to learn of Church's publication. The month after Church's article was published, [Turing 1936] was hurriedly submitted for publication.

[21] The fact that the halting problem is computationally undecidable does not mean that proving that programs halt cannot be done in practice [Cook, Podelski, and Rybalchenko 2006].

Completeness versus Inferential Undecidability

The theorems of a theory $\mathcal{T}$ can be effectively enumerated only in a meta theory of $\mathcal{T}$.

Let **Theorem**$_{\mathcal{T}}$ be a function that effectively enumerates all sentences of $\mathcal{T}$ such that[xvi]

$$\vdash_{\mathcal{T}'} ((\vdash_{\mathcal{T}} \lfloor s \rfloor) \Leftrightarrow \exists i \in \mathbb{N} \rightarrow \mathbf{Theorem}_{\mathcal{T}}(i) = s)$$

$\mathcal{T}'$ (called the meta theory of $\mathcal{T}$) is characterized by the above axiom as an extension of $\mathcal{T}$.[22]

In the case of mathematics, the function is simply called **Theorem** and is a total procedure that satisfies

$$\vdash' ((\vdash \lfloor s \rfloor) \Leftrightarrow \exists i \in \mathbb{N} \rightarrow \mathbf{Theorem}(i) = s)$$

Completeness versus Inferential Undecidability

Computational undecidability implies that mathematics is inferentially undecidable, *i.e.* there is a proposition Ψ such that[23] $\vdash' \nvdash \Psi$ and $\vdash' \nvdash \neg\Psi$

The proposition Ψ is provably inferentially undecidable because it is provably unprovable ($\vdash' \nvdash \Psi$) and its negation is provably unprovable ($\vdash' \nvdash \neg\Psi$).

Information Invariance[24] is a fundamental technical goal of logic consisting of the following:

1. **Soundness of inference:** information is not increased by inference
2. **Completeness of inference:** all information that necessarily holds can be inferred

Direct Logic aims to achieve information invariance even when information is inconsistent using inconsistency robust inference[25] regardless of the proof above that mathematics is inferentially undecidable. As mentioned above, Direct Logic infers there is a proposition Ψ such that both $\vdash' \nvdash \Psi$ *and* $\vdash' \nvdash \neg \Psi$. But this does not mean that it is somehow fundamentally "incomplete" with respect to the information that can be inferred.

Consistency of Mathematics

Consistency can be defined as follows: $\mathsf{Consistent} \equiv \forall s \in \mathrm{Sentences} \rightarrow \nvdash \lfloor s \rfloor, \neg \lfloor s \rfloor$

Theorem: **Mathematics self-proves its own consistency.**[26]

Proof. Suppose to obtain a contradiction that $\neg\mathsf{Consistent}$. Consequently, $\exists s \in \mathrm{Sentences} \rightarrow (\vdash \lfloor s \rfloor)$ and $(\vdash \neg \lfloor s \rfloor)$ and there is a sentence s_0 such that $\vdash \lfloor s_0 \rfloor$ and $\vdash \neg \lfloor s_0 \rfloor$. These theorems can be used to infer $\lfloor s_0 \rfloor$ and $\neg \lfloor s_0 \rfloor$, which is a contradiction.

Using proof by contradiction, $\vdash \mathsf{Consistent}$

[22] $(\vdash \Psi) \Rightarrow' (\vdash' \Psi)$

[23] Proof: For $i \in \mathbb{N}$ let **ProvablyTotal**(i) be an effective enumeration of the provably total procedures. So that

$\vdash' (\forall t \in \mathbf{Terms} \rightarrow (\vdash \mathbf{Total}(t)) \Leftrightarrow \exists i \in \mathbb{N} \rightarrow \mathbf{ProvablyTotal}(i) = t)$

$\mathrm{Diagonal}(i) \equiv \lceil \lfloor \mathbf{ProvablyTotal}(i) \rfloor (i) + 1 \rceil$

By construction $\vdash' \nvdash$ **Total**(Diagonal) because Diagonal differs from every provably total procedure. However, $\vdash'$ **Total**(Diagonal) because **ProvablyTotal**(i) enumerates provably total procedures.

On the other hand, $\vdash' \nvdash \neg$**Total**(Diagonal) because if $\vdash \neg$**Total**(Diagonal) then there is some $i \in \mathbb{N}$ such that $\neg \downarrow \mathrm{Diagonal}(i)$ contradicting $\vdash' \downarrow \mathrm{Diagonal}(i)$

In this way, the inferential undecidability of mathematics is proved *without* using the self-referential propositions of Gödel.

[24] Closely related to conservation laws in physics

[25] See appendices of this paper.

[26] Of course, this is contrary to a famous result in [Gödel1931]. A resolution is that Direct Logic uses a powerful natural deduction system that is its own meta system in the proof below. A tradeoff is that the self-referential propositions of [Gödel1931] (used to prove that mathematics cannot prove its own consistency) are not allowed in Direct Logic. See further discussion in [Hewitt 2012].

Acknowledgment

Important contributions to the semantics of Actors have been made by: Gul Agha, Beppe Attardi, Henry Baker, Will Clinger, Irene Greif, Carl Manning, Ian Mason, Ugo Montanari, Maria Simi, Scott Smith, Carolyn Talcott, Prasanna Thati, and Aki Yonezawa.

Important contributions to the implementation of Actors have been made by: Bill Athas, Russ Atkinson, Beppe Attardi, Henry Baker, Gerry Barber, Peter Bishop, Nanette Boden, Jean-Pierre Briot, Blaine Garst, Bill Dally, Peter de Jong, Jessie Dedecker, Ken Kahn, Bill Kornfeld, Henry Lieberman, Carl Manning, Mark S. Miller, Tom Reinhardt, Chuck Seitz, Richard Steiger, Dale Schumacher, Dan Theriault, Mario Tokoro, Darrell Woelk, and Carlos Varela.

Research on the Actor model has been carried out at Caltech Computer Science, Kyoto University Tokoro Laboratory, MCC, MIT Artificial Intelligence Laboratory, SRI, Stanford University, University of Illinois at Urbana-Champaign Open Systems Laboratory, Pierre and Marie Curie University (University of Paris 6), University of Pisa, University of Tokyo Yonezawa Laboratory and elsewhere.

Conversations over the years with Dennis Allison, Bruce Anderson, Arvind, Bob Balzer, Bruce Baumgart, Gordon Bell, Dan Bobrow, Rod Burstall, Luca Cardelli, Vint Cerf, Keith Clark, Douglas Crockford, Ole-Johan Dahl, Julian Davies, Jack Dennis, Peter Deutsch, Edsger Dijkstra, Scott Fahlman, Dan Friedman, Ole-Johan Dahl, Julian Davies, Patrick Dussud, Doug Englebart, Sol Feferman, Bob Filman, Kazuhiro Fuchi, Mike Genesereth, Cordell Green, Jim Gray, Pat Hayes, Anders Hejlsberg, Pat Helland, John Hennessy, Tony Hoare, Mike Huhns, Dan Ingalls, Anita Jones, Bob Kahn, Gilles Kahn, Alan Karp, Alan Kay, Bob Kowalski, Monica Lam, Butler Lampson, Leslie Lamport, Peter Landin, Vic Lesser, Jerry Lettvin, Lick Licklider, Barbara Liskov, John McCarthy, Dave McQueen, Erik Meijer, Robin Milner, Marvin Minsky, Fanya Montalvo, Ike Nassi, Alan Newell, Kristen Nygaard, Seymour Papert, David Patterson, Carl Petri, Gordon Plotkin, Vaughan Pratt, John Reynolds, Jeff Rulifson, Earl Sacerdoti, Vijay Saraswat, Herbert Simon, Munindar Singh, Dana Scott, Ehud Shapiro, Burton Smith, Guy Steele, Gerry Sussman, Chuck Thacker, Kazunori Ueda, Dave Unger, Richard Waldinger, Peter Wegner, Richard Weyhrauch, Jeannette Wing, Terry Winograd, Glynn Winskel, David Wise, Bill Wulf, *etc.* greatly contributed to the development of the ideas in this article.

Jeremy Forth and Richard Waldinger made very helpful comments and suggestions. Discussions with Dennis Allison, Ron Dolin, Eugene Miya, Vaughan Pratt and others were helpful in improving this article. Pankaj Mehra and Dale Schumacher corrected some typos.

Bibliography

Anthony Anderson and Michael Zelëny (editors). *Logic, Meaning and Computation: Essays in Memory of Alonzo Church* Springer. 2002.
Henry Baker and Carl Hewitt *The Incremental Garbage Collection of Processes* Proceeding of the Symposium on Artificial Intelligence Programming Languages. 1977.
Luca Cardelli and Andrew Gordon. *Mobile Ambients* FoSSaCS'98.
Alonzo Church *The Calculi of Lambda-Conversion* Princeton University Press. 1941.
Alonzo Church. *An unsolvable problem of elementary number theory* American Journal of Mathematics, 58 (1936),
Alonzo Church and J. Barkley Rosser. *Some properties of conversion* Transactions of the American Mathematical Society. May 1936.
Will Clinger. *Foundations of Actor Semantics* MIT Mathematics Doctoral Dissertation. June 1981.
Jack Copeland. *The Essential Turing* Oxford University Press. 2004.
Byron Cook., Andreas Podelski, and Andrey Rybalchenko.. *Termination proofs for systems code*. PLDI. 2006.
Haskell Curry "Some Aspects of the Problem of Mathematical Rigor" *Bulletin of the American Mathematical Society* Vol. 4. 1941.
Ole-Johan Dahl and Kristen Nygaard. *Class and subclass declarations* IFIP TC2 Conference on Simulation Programming Languages. May 1967.
John Dawson *Logical Dilemmas. The Life and Work of Kurt Gödel* AK Peters. 1997
John Dawson. *What Hath Gödel Wrought?* Synthese. Jan. 1998.
John Dawson. *Shaken Foundations or Groundbreaking Realignment? A Centennial Assessment of Kurt Gödel's Impact on Logic, Mathematics, and Computer Science* FLOC'06.
Richard Dedekind (1888) *What are and what should the numbers be?* (Translation in From Kant to Hilbert: A Source Book in the Foundations of Mathematics. Oxford University Press. 1996) Braunschweig.
Liesbeth De Mol. *Generating, solving and the mathematics of Homo Sapiens. Emil Post's views on computation* A Computable Universe: Understanding Computation & Exploring Nature as Computation. Dedicated to the memory

of Alan M. Turing on the 100 th anniversary of his birth. Edited by Hector Zenil. World Scientific Publishing Company. 2012
Cora Diamond. *Wittgenstein's Lectures on the Foundations of Mathematics, Cambridge, 1939* Cornell University Press. 1967.
Edsger Dijkstra. *A Discipline of Programming.* Prentice Hall. 1976.
Edsger Dijkstra and A.J.M. van Gasteren. *A Simple Fixpoint Argument Without the Restriction of Continuity* Acta Informatica. Vol. 23. 1986.
Edsger Dijkstra. *Position Paper on "Fairness"* EWD 1013. E.W. Dijkstra Archive. UT Austin.
T. S. Eliot. *Four Quartets.* Harcourt. 1943.
Solomon Feferman, John Dawson, Stephen Kleene, *et. al.,* editors. *Collected Works of Kurt Gödel Vol. I-V* Oxford University Press. 2001-2003.
Solomon Feferman *Axioms for determinateness and truth* Review of Symbolic Logic. 2008.
Solomon Feferman. *About and around computing over the* reals Computability: Gödel, Church, Turing and Beyond MIT Press. forthcoming 2012.
W.H.J. Feijen, A.J.M. van Gasteren, David Gries and J. Misra (editors). *Beauty is Our Business: Birthday Salute to Edsger W.Dijkstra* Springer. 1990.
Frederic Fitch. *Symbolic Logic: an Introduction*. Ronald Press. 1952.
Nissim Francez, Tony Hoare, Daniel Lehmann, and Willem-Paul de Roever. *Semantics of nondeterminism, concurrency, and communication* Journal of Computer and System Sciences. December 1979.
Gottlob Frege. *Begriffsschrift: eine der arithmetischen nachgebildete Formelsprache des reinen Denkens* Halle, 1879.
John Kenneth Galbraith. *Economics, Peace and Laughter*. New American Library .1971.
Robin Gandy. *Church's Thesis and Principles of Mechanisms* The Kleene Symposium. North–Holland. 1980.
Kurt Gödel (1931) "On formally undecidable propositions of *Principia Mathematica*" in *A Source Book in Mathematical Logic, 1879-1931*. Translated by Jean van Heijenoort. Harvard Univ. Press. 1967.
Irene Greif. *Semantics of Communicating Parallel Professes* MIT EECS Doctoral Dissertation. August 1975
Carl Hewitt and Russ Atkinson. *Specification and Proof Techniques for Serializers* IEEE Journal on Software Engineering. January 1979.
Carl Hewitt, Peter Bishop and Richard Steiger. *A Universal Modular Actor Formalism for Artificial Intelligence* IJCAI-1973.
Carl Hewitt. *What is Commitment? Physical, Organizational, and Social* COIN@AAMAS'06. (Revised version in Springer Verlag Lecture Notes in Artificial Intelligence. Edited by Javier Vázquez-Salceda and Pablo Noriega. 2007) April 2006.
Carl Hewitt *Middle History of Logic Programming: Resolution, Planner, Edinburgh LCF, Prolog, and the Japanese Fifth Generation Project* ArXiv 0904.3036. 2009-2011.
Carl Hewitt *Formalizing common sense for inconsistency-robust information integration using Direct Logic™ Reasoning and the Actor Model* Inconsistency Robustness 2011.
Carl Hewitt *ActorScript™ extension of C#™, Java™ , JavaScript™ and Objective C™* ArXiv. 1008.2748.
Carl Hewitt *Actor Model of Computation: Many-core Inconsistency-robust Information Integration* Inconsistency Robustness 2011.
Carl Hewitt *Mathematics self-proves its own consistency (contra Gödel et. al.)* Submitted for publication to arXiv on March 22, 2012. http://consistency.carlhewitt.info
Tony Hoare. *Communicating Sequential Processes* CACM August, 1978.
Tony Hoare *Communicating Sequential Processes* Prentice Hall. 1985.
Douglas Hofstadter. *Gödel, Escher, Bach: An Eternal Golden Braid* Vintage. 1980.
Cliff Jones, Bill Roscoe, and Kenneth Wood (eds.) *Reflections on the Work of C.A.R. Hoare* Springer. 2010.
Frederick Knabe. *A Distributed Protocol for Channel-Based Communication with Choice* PARLE'92.
Robert Kowalski *The Early Years of Logic Programming* CACM. January 1988.
John Law. *After Method: mess in social science research* Routledge. 2004.
Philip Lewis. *Jonathon von Neumann and EDVAC* Nov. 8. 2004. www.cs.berkeley.edu/~christos/classics/paper.pdf
Martin Löb. *Solution of a problem of Leon Henkin*. Journal of Symbolic Logic. Vol. 20. 1955.
Mark S. Miller. *Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control* Doctoral dissertation. John Hopkins. 2006.
Ray Monk. *Bourgeois, Boshevist or anarchist? The Reception of Wittgenstein's Philosophy of Mathematics* in Wittgenstein and his interpreters. Blackwell. 2007.
David Park. *Concurrency and Automata on Infinite Sequences* Lecture Notes in Computer Science, Vol. 104. Springer. 1980.
Giuseppe Peano *Arithmetices principia, nova methodo exposita* (The principles of arithmetic, presented by a new method) 1889.
Gordon Plotkin. *A powerdomain construction* SIAM Journal of Computing September 1976.
Gordon Plotkin, Colin Stirling and Mads Tofte (editors). *Proof, Language, and Interaction: Essays in Honour of Robin Milner* MIT Press. 2000.
Gordon Plotkin. *Robin Milner: A Craftsman of Tools for the Mind* YouTube. 2010.
Robin Milner. *Elements of Interaction* CACM. January 1993.
Bill Roscoe. *The Theory and Practice of Concurrency* Prentice-Hall. Revised 2005.
Bertrand Russell. *Principles of Mathematics* Norton. 1903.
Bertrand Russell. *Principia Mathematica 2nd Edition* 1925.

Dana Scott *Data Types as Lattices*. SIAM Journal on computing. 1976.
Oron Shagrir *Gödel on Turing on Computability* Church's Thesis after 70 years Ontos-Verlag. 2006.
Wilfried Sieg and J. Byrnes *An Abstract Model for Parallel Computations: Gandy's Thesis* Monist. 1999.
Wilfried Sieg *Church Without Dogma – axioms for computability* New Computational Paradigms. Springer Verlag. 2008.
Alan Turing. *On computable numbers, with an application to the Entscheidungsproblem* Proceedings London Math Society. 1936.
Alan Turing. *Intelligent Machinery* National Physical Laboratory Report. 1948.
Hao Wang *A Logical Journey, From Gödel to Philosophy* MIT Press. 1974.
Ludwig Wittgenstein. 1956. *Bemerkungen ¨uber die Grundlagen der Mathematik*/*Remarks on the Foundations of Mathematics, Revised Edition* Basil Blackwell. 1978
Ludwig Wittgenstein. *Philosophische Grammatik* Basil Blackwell. 1969.
Ludwig Wittgenstein. (1933-1935) *Blue and Brown Books.* Harper. 1965.
Ludwig Wittgenstein *Philosophical Investigations* Blackwell. 1953/2001.

Appendix 1. Historical development

"Faced with the choice between changing one's mind and proving that there is no need to do so, almost everyone gets busy on the proof."
John Kenneth Galbraith [1971 pg. 50]

Truth versus Argumentation

Mathematics progressed by characterizing structures up to isomorphism including the integers [Peano 1889] and the real numbers up to isomorphism [Dedekind 1888] with the following theorems:

- *Full Peano Arithmetic:* Let **X** be the structure $<X, 0_X, S_X>$, then
  $\text{Peano}[\mathbf{X}] \Rightarrow \mathbf{X} \approx <\mathbb{N}, 0, S>$
  where $\text{Peano}[X, 0_X, S_X]$, means that **X** satisfies the full Peano axioms for the non-negative integers, $\mathbb{N}$[xvii] is the set of non-negative integers, S is the successor function, and $\approx$ means isomorphism.[xviii] Note that the theory *Peano* is full Peano arithmetic that is strictly more powerful than cut-down first-order arithmetic where induction is limited to only first-order expressions.[xix] For example, there are nondeterministic Turing machines that *Peano* proves always halt that cannot be proved to halt in the cut-down first-order version of arithmetic.
- *Real Numbers:* Let **X** be the structure $<X, \leqq_X, 0_X, 1_X, +_X, *_X>$, then[27]
  - $\text{Dedekind}[\mathbf{X}] \Rightarrow \mathbf{X} \approx <\mathbb{R}, \leqq, 0, 1, +, *>$[28]
  - $\text{Cauchy}[\mathbf{X}] \Rightarrow \mathbf{X} \approx <\mathbb{R}, \leqq, 0, 1, +, *>$[29]

Principia Mathematica [Russell 1925] (denoted by the theory *Russell*) was intended to be a foundation for all of mathematics including Set Theory and Analysis building on [Frege 1879].

However, using a self-referential set,[30] Russell discovered an inconsistency. The inconsistency was resolved by placing restrictions on the construction of sets to prevent the construction of Russell's self-referential set. In the restricted set theory, [Zermelo 1930] proved that or any two models of ZFC^2 (2nd order Zermelo-Fraenkel with Choice axioms for set theory) one of them is isomorphically embeddable in the other.

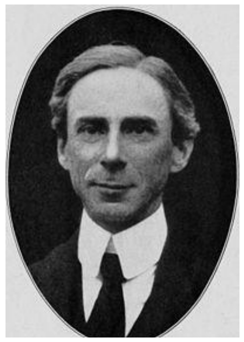
The above results categorically characterize mathematical structures based on *argumentation*. There is no way to go beyond argumentation to get at some special added insight called "*truth*." Argumentation is all that we have. For example, the only way to know that proposition is "*true*" of the Peano numbers is to prove the proposition from the Peano axioms.

Giuseppe Peano

Richard Dedekind

Augustin-Louis Cauchy

Gottlob Frege


Bertrand Russell

Ernst Zermelo

[27] $\mathbb{R}$ is the set of real numbers, and $\approx$ means isomorphism.

[28] Dedekind[**X**], means that **X** satisfies the Dedekind axioms for the real numbers

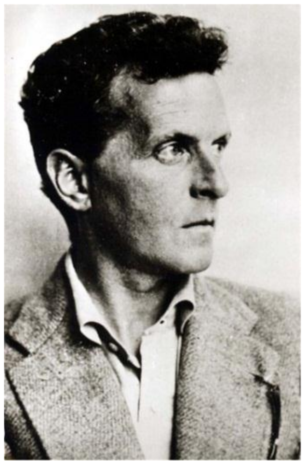
[29] Cauchy[**X**], means that **X** satisfies the Cauchy axioms for the real numbers

[30] Namely, the set of all set that are not members of themselves

Turing versus Wittgenstein

Turing differed fundamentally on the question of inconsistency from Wittgenstein when he attended Wittgenstein's seminar on the Foundations of Mathematics [Diamond 1976]:

> *Wittgenstein:... Think of the case of the Liar. It is very queer in a way that this should have puzzled anyone — much more extraordinary than you might think... Because the thing works like this: if a man says 'I am lying' we say that it follows that he is not lying, from which it follows that he is lying and so on. Well, so what? You can go on like that until you are black in the face. Why not? It doesn't matter. ...it is just a useless language-game, and why should anyone be excited?*
>
> *Turing: What puzzles one is that one usually uses a contradiction as a criterion for having done something wrong. But in this case one cannot find anything done wrong.*
>
> *Wittgenstein: Yes — and more: nothing has been done wrong, ... where will the harm come?*
>
> *Turing: The real harm will not come in unless there is an application, in which a bridge may fall down or something of that sort…. You cannot be confident about applying your calculus until you know that there are no hidden contradictions in it*[31]*…. Although you do not know that the bridge will fall if there are no contradictions, yet it is almost certain that if there are contradictions it will go wrong somewhere.*[32]


Ludwig Wittgenstein

Wittgenstein followed this up with [Wittgenstein 1956, pp. 104e–106e]: *Can we say: 'Contradiction is harmless if it can be sealed off'? But what prevents us from sealing it off?*

The above debate between Turing and Wittgenstein has continued to this day with interesting developments described below.

von Neumann on Inferential Undecidability

According to [Lewis 2004]:

> *Von Neumann often had insights into the repercussions of work that others would understand later; on hearing Gödel present his results on formal incompleteness* [inferential undecidability], *he immediately forsook logic and said "**it's all over.**"* (emphasis added)

John von Neumann

From the very beginning, von Neumann strongly disagreed with Gödel's interpretation of inferential undecidability and concluded "*... there is no rigorous justification for classical mathematics*."[xx]

---

[31] Church and Turing later proved that determining whether there are hidden contradictions in a useful calculus is computationally undecidable.

[32] Turing was correct that it is unsafe to use classical logic to reason about inconsistent information. For this reason, inconsistency-robust logic[Hewitt 2011] has been developed to more safely reason about inconsistent information.

Wittgenstein: self-referential propositions lead to inconsistency

Not known, because not looked for T.S. Eliot [1942]

Having previously conceived inconsistency tolerant logic, Wittgenstein had his own interpretation of inferential undecidability (which was completely at odds with Gödel)::[xxi]

> *"True in Russell's system" means, as we have said, proved in Russell's system; and "false in Russell's system" means that the opposite* [negation] *has been proved in Russell's system.*
> *Let us suppose I prove*[33] *the unprovability (in Russell's system* [$\mathcal{Russell}$] *) of P* [$\vdash_{\mathcal{Russell}} \nvdash_{\mathcal{Russell}} P$ where $P \Leftrightarrow \nvdash_{\mathcal{Russell}} P$]; *then by this proof I have proved P* [$\vdash_{\mathcal{Russell}} P$].
> *Now if this proof were one in Russell's system* [$\vdash_{\mathcal{Russell}} \vdash_{\mathcal{Russell}} P$]*—I should in this case have proved at once that it belonged* [$\vdash_{\mathcal{Russell}} P$] *and did not belong* [$\vdash_{\mathcal{Russell}} \neg P$ because $\neg P \Leftrightarrow \vdash_{\mathcal{Russell}} P$] *to Russell's system.*
> *But there is a contradiction here* [in $\mathcal{Russell}$]*!*[34]*—Well, then there is a contradiction here.*

Thus the attempt to develop a universal system of classical mathematical logic[xxii] once again ran into inconsistency. That a theory that infers its own inferential undecidability using the self-referential proposition P is inconsistent represented a huge threat to Gödel's firmly held belief that mathematics is based on objective truth.[xxiii]

Classical logicians versus Wittgenstein

> *The powerful (try to) insist that their statements are literal depictions of a single reality. 'It really is that way', they tell us. 'There is no alternative.' But those on the receiving end of such homilies learn to read them allegorically, these are techniques used by subordinates to read through the words of the powerful to the concealed realities that have produced them.*
> Law [2004]

In 1972, Gödel said: "*Has Wittgenstein lost his mind? Does he mean it seriously?*"[xxiv]

The upshot is that Gödel never acknowledged that inferential undecidability based on his self-referential proposition implies inconsistency. Later, Gödel retreated to using Peano arithmetic with the added assumption of self-referential propositions. However, even this retreat did not evade Wittgenstein's point because the theory $\mathcal{Peano+SelfReference}$ is inconsistent in the same way.

Also, the ultimate criteria for correctness in the theory of arithmetic is *provability* in the theory of 2nd order logic with general induction [Dedekind 1888, Peano 1889]. In this sense, Wittgenstein was correct in his identification of "truth" with provability. On the other hand, **Gödel obfuscated the important identification of provability as the touchstone of ultimate correctness in mathematics.**

---

33 Wittgenstein was granting the supposition that Gödel had proved inferential undecidability in Russell's system, *e.g.*
$\vdash_{\mathcal{Russell}} \nvdash_{\mathcal{Russell}} P$. However, inferential undecidability is easy to prove using Gödel's self-referential proposition. Suppose to obtain a contradiction that $\vdash_{\mathcal{Russell}} P$. Both of the following can be inferred:
1) $\vdash_{\mathcal{Russell}} \nvdash_{\mathcal{Russell}} P$ from the hypothesis because $P \Leftrightarrow \nvdash_{\mathcal{Russell}} P$
2) $\vdash_{\mathcal{Russell}} \vdash_{\mathcal{Russell}} P$ from the hypothesis by Adequacy.

But 1) and 2) are a contradiction in $\mathcal{Russell}$. Consequently, $\vdash_{\mathcal{Russell}} \nvdash_{\mathcal{Russell}} P$ follows from proof by contradiction in $\mathcal{Russell}$.

34 Wittgenstein was saying that Gödel's self-referential proposition *P* shows that Russell's system is inconsistent in much the same way that Russell had previously shown Frege's system to be inconsistent using the self-referential set of all sets that are not members of themselves.

Gödel responded as follows to Wittgenstein:[xxv]

> *He* [Wittgenstein] *has to take a position when he has no business to do so. For example, "you can't derive everything from a contradiction." He should try to develop a system of logic in which that is true.*[35]

According to [Monk 2007]:

> *Wittgenstein hoped that his work on mathematics would have a cultural impact, that it would threaten the attitudes that prevail in logic, mathematics and the philosophies of them. On this measure it has been a spectacular failure.*

Unfortunately, recognition of the worth of Wittgenstein's work on mathematics came long after his death. Classical logicians mistakenly believed that they had been completely victorious over Wittgenstein. For example, according to [Dawson 2006 *emphasis in original*]:

- *Gödel's results altered the mathematical landscape, but they did* **not** *"produce a debacle".*
- *There is* **less** *controversy today over mathematical foundations than there was* **before** *Gödel's work.*

However, the groundbreaking realignment came later when computer science invented a useable inconsistency robust logic because of pervasive inconsistency in computer information systems.

The controversy between Wittgenstein and Gödel can be summarized as follows:

- Gödel
  1. Mathematics is based on objective truth.[36]
  2. A theory is not allowed to *directly* reason about itself.
  3. Self-referential sentences prove inferential undecidability but (hopefully) not inconsistency.
  4. Theories should be proved consistent.
- Wittgenstein
  1. Mathematics is based on communities of practice.
  2. Reasoning about theories is like reasoning about everything else, *e.g.* chess.
  3. Self-referential sentences can lead to inconsistency.
  4. Theories should use inconsistency robust inference.

According to Feferman [2008]:

> *So far as I know, it has not been determined whether such* [inconsistency robust] *logics account for "sustained ordinary reasoning", not only in everyday discourse but also in mathematics and the sciences.*

Direct Logic [Hewitt 2011] was put forward as an improvement over classical logic with respect to Feferman's desideratum above. Computer science needs an all-embracing system of inconsistency-robust reasoning to implement practical information integration.[37]

Turing versus Gödel

> *You shall not cease from exploration*
> *And the end of all our journeying*
> *Will be to arrive where we started*
> *And know the place for the first time.*
> T.S. Eliot [1942]

Turing recognized that proving inference for $\mathcal{Russell}$ is computationally undecidable is quite different than proposing that a self-referential proposition proves $\mathcal{Russell}$ is inferentially undecidable[Turing 1936, page 259]:

> It should perhaps be remarked what I shall prove is quite different from the well-known results of Gödel [1931]. Gödel has shown that (in the formalism of Principia Mathematica) there are propositions U such that neither U nor ¬U is provable. … On the other hand, I shall show that there is no general method which tells whether a given formula U is provable[xxvi]

[35] Gödel knew that it would be technically difficult to develop a useful system of logic proposed by Wittgenstein in which "*you can't derive everything from a contradiction*" and evidently doubted that it could be done.

[36] According to [Gödel 1951 page 30] mathematical objects and "*concepts form an objective reality of their own, which we cannot create or change, but only perceive and describe.*"

[37] Computer systems need all-embracing rules to justify their inferences, *i.e.*, they can't always rely on human manual intervention.

The proof method of showing computational undecidability developed by Turing and Church proves that *Russell* is inferentially undecidable without using self-referential propositions.

Although they share some similar underlying ideas, the method of proving inferential undecidability developed by Church and Turing is much more robust than the one previously developed by Gödel that relies on self-referential sentences. The difference can be explicated as follows:

- Actors: an Actor that has an address for itself can be used to generate infinite computations.
- Sentences: a sentence that has a reference to itself can be used to infer inconsistencies.

As Wittgenstein pointed out, the self-referential "*This sentence is not provable*" leads an inconsistency in the foundations of mathematics. If the inconsistencies of self-referential propositions stopped with this example, then it would be somewhat tolerable for an inconsistency-robust theory. However, other self-referential propositions (constructed in a similar way) can be used to prove every proposition thereby rendering inference useless.[xxvii]

This is why Direct Logic does not support self-referential propositions. The standard way to construct self-referential sentences does not work.[38]

λ-definability versus Turing machines

That Actors which behave like mathematical functions exactly correspond with those definable in the lambda calculus provides an intuitive justification for the rules of the lambda calculus.

According to [De Mol 2012]:

> *But why exactly is Turing's thesis so much more appealing than Church's? This issue … comes down to the fact that unlike Church, Turing's thesis makes a "direct appeal to intuition"*[39]
>
> *Church only arrived at his thesis after several research results on λ-calculus had been established. It was its equivalence with general recursive functions and the fact that Church, Kleene and Rosser were able to λ-define any function they could come up with, that resulted in Church's formulation of the thesis in terms of λ-calculus and later in terms of general recursiveness. In other words, Church did not start out from the idea of trying to formally capture some intuition, but, on the contrary, only saw that the formalism he was working with might be powerful enough to capture all "calculable" functions after a thorough analysis of that formalism.*
>
> *This explains why defining effective calculability in terms of λ-definability or general recursiveness is not "intuitively appealing" since neither of these symbolic systems was constructed with that purpose in mind.*

The Actor model provides a direct appeal to intuition for λ-definablity that was lacking in its initial formulation.

[38] For example, the fixed point of a propositional p is usually defined as follows:

$$\mathrm{Diagonal}_p \equiv \ulcorner(s\in\mathrm{Sentences})\rightarrow \lfloor p\rfloor_T(\ulcorner\lfloor s\rfloor_T(s)\urcorner_T)\urcorner_T$$

$$\mathrm{Fix}(p) \equiv \lfloor \mathrm{Diagonal}_p\rfloor_T(\mathrm{Diagonal}_p)$$

A problem arises applying the above definition in proving that Fix(p) is a sentence in Direct Logic.

[39] [Turing 1937 pg. 249] (emphasis in source)

Appendix 2. Inconsistency-robust Logic Programming

Logic Programs[40] logically infer computational steps.

$\triangleleft$ "$\vdash$"$_{\text{Theory}}$ PropositionExpression $\triangleright$:: Expression

Assert PropositionExpression for Theory.

Forward Chaining

$\triangleleft$ "**when**" "$\vdash$"$_{\text{Theory}}$ PropositionPattern "$\rightarrow$" Expression $\triangleright$:: Continuation

When PropositionPattern holds for Theory, evaluate Expression*.*

Illustration of forward chaining:
{$\vdash_{t}$ Human[Socrates]; **when** $\vdash_{t}$ Human[x] $\rightarrow$ $\vdash_{t}$ Mortal[x]} will result in asserting Mortal[Socrates] for theory t

Backward Chaining

$\triangleleft$ "?"$_{\text{Theory}}$ GoalPattern "$\rightarrow$" Expression $\triangleright$:: Continuation

Set GoalPattern for Theory and when established evaluate Expression.

$\triangleleft$ "?"$_{\textit{Theory}}$ GoalPattern $\triangleright$:: Expression

Set GoalPattern for Theory and return a list of assertions that satisfy the goal.

$\triangleleft$ "**when**" "?"$_{\textit{Theory}}$ GoalPattern "$\rightarrow$" Expression $\triangleright$:: Continuation

When there is a goal that matches GoalPattern for Theory, evaluate Expression.

Illustration of backward chaining:
{$\vdash_{t}$ Human[Socrates]**, when** $?_{t}$ Mortal[x] $\rightarrow$ ($?_{t}$ Human[=x] $\rightarrow$ $\vdash_{t}$ Mortal[x]), **?**$_{t}$ Mortal[Socrates]**}**
will result in asserting Mortal[Socrates] for theory t.

SubArguments

This section explains how subarguments[41] can be implemented in natural deduction.

when $?_{s}$ (psi $\vdash_{t}$ phi) $\rightarrow$ **let** t' = extension(t) $\rightarrow$ **{**$\vdash_{t'}$ psi**, ?**$_{t'}$ =phi $\rightarrow$ $\vdash_{s}$ (psi $\vdash_{t}$ phi)**}**

Note that the following hold for t' because it is an extension of t:

when $\vdash_{t}$ theta $\rightarrow$ $\vdash_{t'}$ theta;;

when **?**$_{t'}$ theta $\rightarrow$ **?**$_{t}$ theta;;

[40] [Church 1932; McCarthy 1963; Hewitt 1969, 1971, 2010; Milner 1972, Hayes 1973; Kowalski 1973]. Note that this definition of Logic Programming does *not* follow the proposal in [Kowalski 1973, 2011] that Logic Programming be restricted only to backward chaining*, e.g.,* to the exclusion of forward chaining, *etc.*

[41] See appendix on Inconsistency Robust Natural Deduction.

Appendix 3. Inconsistency-robust Natural Deduction

Below are schemas for nested-box-style Natural Deduction [Fitch 1952] for Direct Logic:[42]

$\vdash$ **Introduction (SubArgument)**	
$\vdash_{T\wedge\Psi}\Psi$	ⓘ hypothesis
…	
$\vdash_{T\wedge\Psi}\Phi$	ⓘ inference
$\Psi\vdash_T\Phi$	ⓘ **conclusion**
$(\vdash_{T\wedge\Psi}\Phi)\ \vdash_T(\Psi\vdash_T\Phi)$	

$\vdash$ **Elimination (Chaining)**	
$\vdash_T\Psi$	ⓘ premise
…	
$\Psi\vdash_T\Phi$	ⓘ premise
…	
$\vdash_T\Phi$	ⓘ **conclusion**
$\Psi,(\Psi\vdash_T\Phi)\ \vdash_T\Phi$	

$\wedge$ **Introduction**	
$\vdash_T\Psi$	ⓘ premise
…	
$\vdash_T\Phi$	ⓘ premise
…	
$\vdash_T(\Psi\wedge\Phi)$	ⓘ **conclusion**
$\Psi,\Phi\ \vdash_T(\Psi\wedge\Phi)$	

$\wedge$ **Elimination**	
$\vdash_T(\Psi\wedge\Phi)$	ⓘ premise
…	
$\vdash_T\Psi$	ⓘ **conclusion**
…	
$\vdash_T\Phi$	ⓘ **conclusion**
$(\Psi\wedge\Phi)\ \vdash_T\Psi,\Phi$	

$\vee$ **Introduction**	
$\vdash_T\Psi$	ⓘ premise
…	
$\vdash_T\Phi$	ⓘ premise
…	
$\vdash_T(\Psi\vee\Phi)$	ⓘ **conclusion**
$\Psi,\Phi\ \vdash_T(\Psi\vee\Phi)$	

$\vee$ **Elimination**	
$\vdash_T\neg\Psi$	ⓘ premise
…	
$\vdash_T(\Psi\vee\Phi)$	ⓘ premise
…	
$\vdash_T\Phi$	ⓘ **conclusion**
$\neg\Psi,(\Psi\vee\Phi)\ \vdash_T\Phi$	

$\vee$ **Cases**	
$\vdash_T(\Psi\vee\Phi)$	ⓘ premise
…	
$\Psi\vdash_T\Theta$	ⓘ premise
…	
$\Phi\vdash_T\Omega$	ⓘ premise
…	
$\vdash_T(\Theta\vee\Omega)$	ⓘ **conclusion**
$(\Psi\vee\Phi),(\Psi\vdash_T\Theta),(\Phi\vdash_T\Omega)\ \vdash_T(\Theta\vee\Omega)$	

$\Rightarrow$ **Introduction**	
$\Psi\vdash_T\Phi$	ⓘ premise
…	
$\neg\Phi\vdash_T\neg\Psi$	ⓘ premise
…	
$\Psi\Rightarrow_T\Phi$	ⓘ **conclusion**
$(\Psi\vdash_T\Phi),(\neg\Phi\vdash_T\neg\Psi)\vdash_T(\Psi\Rightarrow_T\Phi)$	

$\Rightarrow$ **Elimination**	
$\Psi\Rightarrow_T\Phi$	ⓘ premise
…	
$\Psi\vdash_T\Phi$	ⓘ **conclusion**
…	
$\neg\Phi\vdash_T\neg\Psi$	ⓘ **conclusion**
$(\Psi\Rightarrow_T\Phi)\ \vdash_T(\Psi\vdash_T\Phi),(\neg\Phi\vdash_T\neg\Psi)$	

Integrity
$(\vdash_T\Psi)\Rightarrow_T\Psi$

Reflection (Adequacy and Faithfulness)
$(\Phi\vdash_T\Psi)\Leftrightarrow_T(\vdash_T(\Phi\vdash_T\Psi))$

See the section on Inconsistency-robust Logic Programming for how this can be implemented

[42] In addition to the usual Boolean equivalences.

End Notes

[i] See [Gandy 1980] and [Sieg 1999, 2008] for further development of Turing's model of computation.

[ii] To first approximation in the Actor Model, we have the following:

- a partial order (called the "Activation Order") on events that activate other events (This ordering is a generalization of the one in the parallel lambda calculus.)
- a separate total order for each Actor (called its "Reception Order") on events with messages received by the Actor

These orderings are not part of the parallel lambda calculus. The axioms for Actor Systems [Baker and Hewitt 1977, Hewitt 2010] stated required relationships among these orderings. Since the Actor Model, rejected the Global State Assumption of previous models of computations, there is no identification of events with global states.

[iii] A nondeterministic system is defined to have unbounded nondeterminism exactly when both of the following hold:

1. When started, the system always halts.
2. For every integer n, it is possible for the system to halt with an output that is greater than n.

For example the following systems do *not* have unbounded nondeterminism:

- A nondeterministic system which sometimes halts and sometimes doesn't
- A nondeterministic system that always halts with an output less than 100,000.
- An operating system that never halts

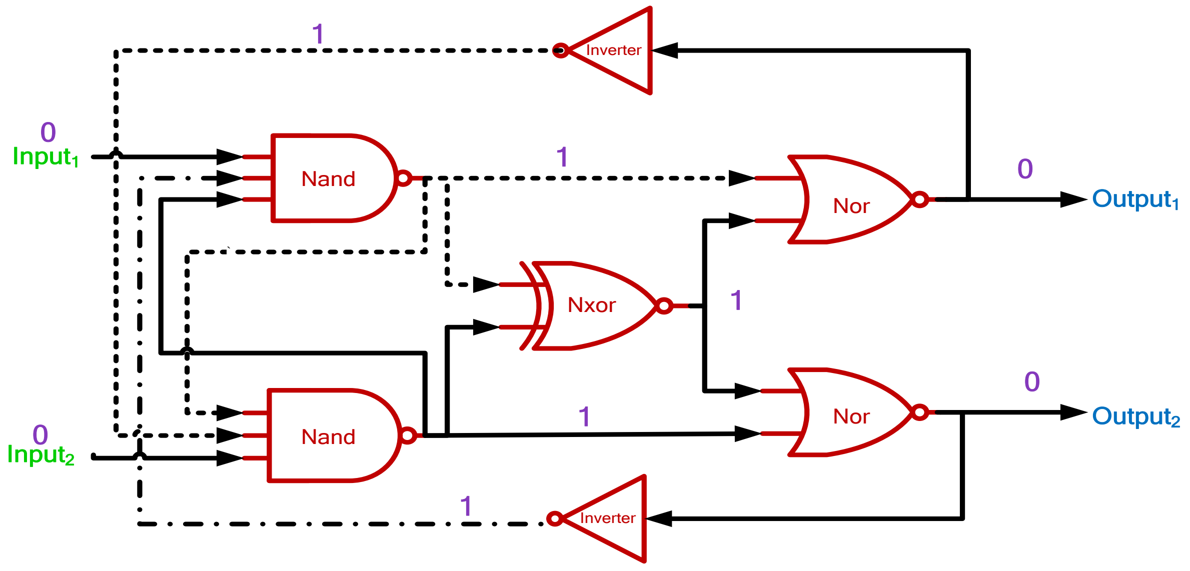
[iv] This result is very old. It was known by Dijkstra motivating his belief that it is impossible to implement unbounded nondeterminism. Also the result played a crucial role in the invention of the Actor Model in 1972. The proof also applies to the Abstract State Machine (ASM) model [Blass, Gurevich, Rosenzweig, and Rossman 2007a, 2007b; Glausch and Reisig 2006].

[v] Consequently, it does not implement unbounded nondeterminism. Some people have argued that it is "unfair" for the nondeterministic Turing machine to always make the first choice in Step 1. This had led to a confusing body of literature on various kinds on "unfairness." For example, if a nondeterministic program makes choices in two different steps in the program is it "unfair" for them to be correlated? [Dijkstra 1987] argued against "unfairness" because it cannot be proved by any finite computation.

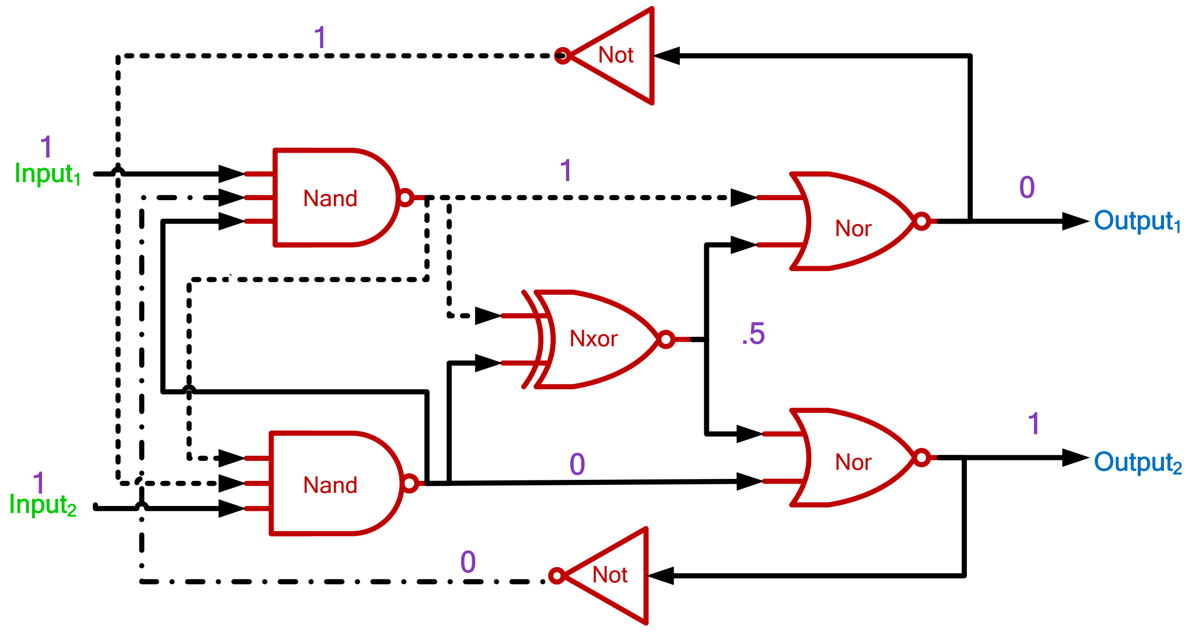
[vi] An example of the global state model is the Abstract State Machine (ASM) model [Blass, Gurevich, Rosenzweig, and Rossman 2007a, 2007b; Glausch and Reisig 2006].

[vii] *cf.* denotational semantics of the lambda calculus [Scott 1976]

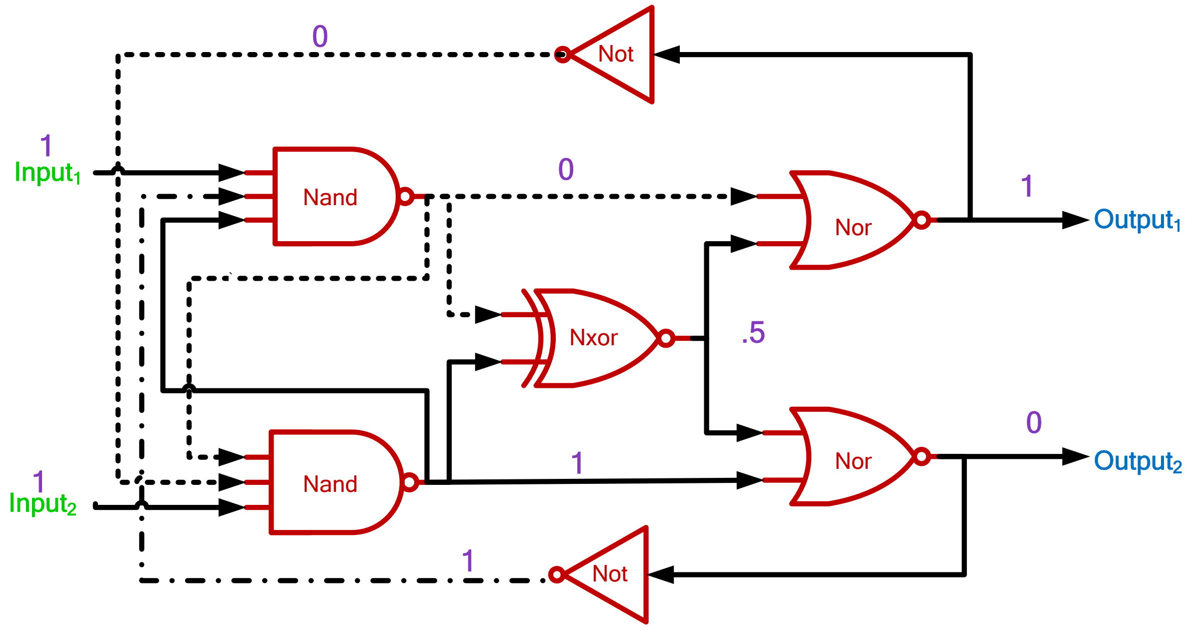
[viii] Example below show the outcome of Input_1=0 and Input_2=0:



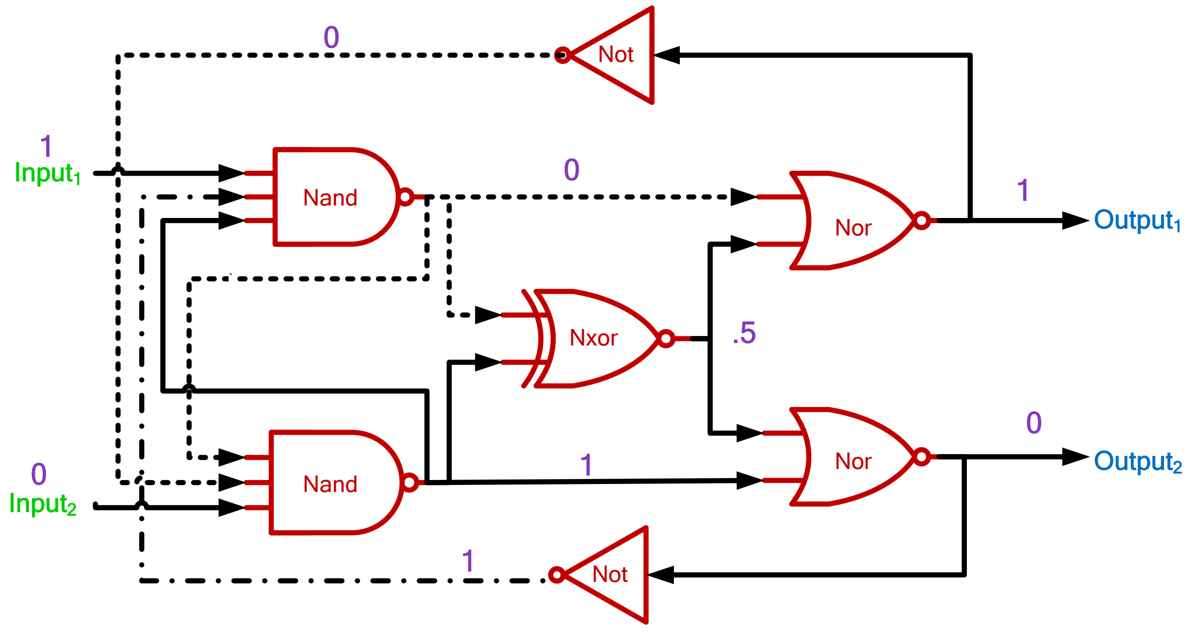
Example below shows a possible outcome of Input$_1$=1 and Input$_2$=1:



Example below shows the other possible outcome of Input$_1$=1 and Input$_2$=1:



Example below shows the outcome of Input$_1$=1 and Input$_2$=0:



[ix] Arbiters render meaningless the states in the Abstract State Machine (ASM) model [Blass, Gurevich, Rosenzweig, and Rossman 2007a, 2007b; Glausch and Reisig 2006].

[x] See [Feijen, van Gasteren, Gries and Misra 1990] on the work of Dijkstra.

[xi] See [Roscoe 2005].

[xii] Furthermore, a Logic Program cannot implement unbounded nondeterminism where a Logic Program is defined by the criteria that it must logically infer its computational steps. See discussion in [Hewitt ArXiv 0812.4852].

[xiii] See [Knabe 1992].

[xiv] See [Plotkin, Stirling and Tofte 2000] and [Plotkin 2010] on Milner's work.

[xv] Proof: Suppose to obtain a contraction that ComputationallyDecidable[HaltingProblem].
Define a procedure Diagonal as follows: Diagonal ≡⌈(x) → Halt(x, x) **??** (*True* → ↑( )⸱*False* → *True*)⌉
Poof of inconsistency: By the definition of Diagonal:
⌊Diagonal⌋ (Diagonal) $\rightarrow_1$ Halt(Diagonal, Diagonal) **??** (*True* → ↑( ), *False* →*True*)
Consider the following 2 cases:

1. Halt(Diagonal, Diagonal) $\rightarrow_1$ *True*
   $\underline{\downarrow}$(⌊Diagonal⌋(Diagonal)) by the axioms for Halt
   ¬$\underline{\downarrow}$(⌊Diagonal⌋(Diagonal)) by the definition of Diagonal
2. Halt(Diagonal, Diagonal) $\rightarrow_1$ *False*
   ¬$\underline{\downarrow}$(⌊Diagonal⌋(Diagonal)) by the axioms for Halt
   $\underline{\downarrow}$(⌊Diagonal⌋(Diagonal)) by the definition of Diagonal

Consequently, ¬ComputationallyDecidable[HaltingProblem]

[xvi] Note:

- **Theorem**$_{\mathcal{T}}$ may be indeterminate, *e.g., $\mathcal{T}$* may have a live data feed that will produce more results in the future.
- **Theorem**$_{\mathcal{T}}$ may be uncomputable, e.g., the axioms of $\mathcal{T}$ may not be effectively enumerable because may have a live data feed that will produce more results in the future.

[xvii] $\mathbb{N}$ is defined as follows:

- $0\in \mathbb{N}$
- $\forall n\in\mathbb{N}\rightarrow\ S(n)\in\mathbb{N}$
- $\forall n\in\mathbb{N}\rightarrow\ S(n)\neq 0$
- $\forall n\in\mathbb{N}\rightarrow\ S(n)=S(m) \Rightarrow n=m$
- $\forall Y\rightarrow$ Inductive[Y]$\Rightarrow \mathbb{N}\subseteq Y$

  where Inductive[Y] is defined as follows:
  - $0_X\in Y$
  - $\forall n\in Y\rightarrow S(n)\in Y$

[xviii] The isomorphism is proved by defining a function f from $\mathbb{N}$ to X by:

1. $f(0)=0_X$
2. $f(S(n))=S_X(f(n))$

Using proof by induction, the following follow:

1. f is defined for every element of $\mathbb{N}$
2. f is one-to-one

   Proof:

   First prove $\forall n\in X\rightarrow f(n)=0_X \Rightarrow n=0$

   Base: Trivial.

   Induction: Suppose $f(n)=0_X \Rightarrow n=0$

   $f(S(n))=S_X(f(n))$ Therefore if $f(S(n))=0_X$ then $0_X=S_X(f(n))$ which is an inconsistency

   Suppose $f(n)=f(m)$. To prove: $n=m$

   Proof: By induction on n:

   Base: Suppose $f(0)=f(m)$. Then $f(m)= 0_X$ and $m=0$ by above

   Induction: Suppose $\forall m\in\mathbb{N}\rightarrow f(n)=f(m)\Rightarrow n=m$

   Proof: By induction on m:

   Base: Suppose $f(n)=f(0)$. Then $n=m=0$

   Induction:

   Suppose $f(n)=f(m)\Rightarrow n=m$

   $f(S(n))=S_X(f(n))$ and $f(S(m))=S_X(f(m))$

   Therefore $f(S(n))=f(S(m))\Rightarrow S(n)=S(m)$

3. the range of f is all of X.

   Proof: To show: Inductive[Range(f)]

   Base: To show $0_X\in$Range(f). Clearly $f(0)=0_X$

   Induction: To show $\forall n\in$Range(f) $\rightarrow$ $S_X(n)\in$Range(f).

   Suppose that $n\in$Range(f). Then there is some m such that $f(m)=n$.

   To prove: $\forall k\in\mathbb{N}\rightarrow f(k)=n \Rightarrow S_X(n)\in$Range(f)

   Proof: By induction on k:

   Base: Suppose $f(0)=n$. Then $n= 0_X =f(0)$ and $S_X(n)=f(S(0))\in$Range(f)

   Induction: Suppose $f(k)=n \Rightarrow S_X(n)\in$Range(f)

   Suppose $f(S(k))=n$. Then $n=S_X(f(k))$ and

   $S_X(n)=S_X(S_X(f(k)))=S_X(f(S(k)))= f(S(S(k)))\in$Range(f)

[xix] [Gentzen 1942] proved the consistency of the cut-down first-order version of arithmetic.

[xx] letter of von Neumann to Gödel, November 29, 1930

[xxi] [Wittgenstein 1956, p. 50e and p. 51e]

[xxii] beginning with Frege [1893]

[xxiii] Gödel was a follower of the school of mathematics founded primarily by Hilbert and his fellow workers. Bois-Reymond [1872] had expressed skepticism with the Latin maxim "*ignoramus et ignorabimus*" [we do not know and will not know]. Hilbert [1900] responded as follows: *that within us we always hear the call: here is the problem, search for the solution: you can find it by pure thought, for in mathematics there is no* ***ignorabimus****.*

Paul Cohen [2006] wrote as follows of his interaction with Gödel:

His [Gödel's] *main interest seemed to lie in discussing the "truth" or "falsity" of these questions, not merely in their undecidability. He struck me as having an almost unshakable belief in this "realist" position, which I found difficult to share. His ideas were grounded in a deep philosophical belief as to what the human mind could achieve. I greatly admired this faith in the power and beauty of Western Culture, as he put it, and would have liked to understand more deeply what were the sources of his strongly held beliefs. Through our discussions, I* *came closer to his point of view, although I never shared completely his "realist" point of view, that all questions of Set Theory were in the final analysis, either true or false.*

In contrast, von Neumann [1961] came to very different conclusions:

*It is **not** necessarily true that the mathematical method is something absolute, which was revealed from on high, or which somehow, after we got hold of it, was evidently right and has stayed evidently right ever since.*

xxiv [Wang 1997] pg. 197.

xxv [Gödel in 5 April 1972 letter to Carl Menger quoted in Wang 1997]

xxvi The inferability problem is to computationally decide whether a proposition defined by sentence is inferable.
Theorem [Church 1936, Turing 1936]. $\text{Consistent}_{T} \Rightarrow \neg\text{ComputationallyDecidable}[\text{InferenceProblem}_{T}]$

Proof. Suppose to obtain a contradiction that $\text{ComputationallyDecidable}[\text{InferenceProblem}_{T}]$.
This means that there is a total computational deterministic predicate Inferable_{T} such that the following 3 properties hold

1. $\text{Inferable}_{T}(s) \rightarrow_1 \mathit{True} \Leftrightarrow \vdash_{T} \lfloor s \rfloor_{T}$
2. $\text{Inferable}_{T}(s) \rightarrow_1 \mathit{False} \Leftrightarrow \nvdash_{T} \lfloor s \rfloor_{T}$
3. $\text{Inferable}_{T}(s) \rightarrow_1 \mathit{True} \vee \text{Inferable}_{T}(s) \rightarrow_1 \mathit{False}$

The proof proceeds by showing that if inference is computationally decidable, the halting problem is computationally decidable. Consider sentences of the form $\lceil \underline{\downarrow}(\lfloor p \rfloor(x)) \rceil$, which is the sentence that the program p halts on input x.

Lemma: $\text{Consistent}_{T} \Rightarrow \text{Inferable}_{T}(\lceil \underline{\downarrow}(\lfloor p \rfloor(x)) \rceil) \rightarrow_1 \mathit{True}$ if and only if $\underline{\downarrow}(\lfloor p \rfloor(x))$

Proof of lemma: Suppose Consistent_{T}

1. Suppose $\text{Inferable}_{T}(\lceil \underline{\downarrow}(\lfloor p \rfloor(x)) \rceil) \rightarrow_1 \mathit{True}$. Then $\vdash_{T} \underline{\downarrow}(\lfloor p \rfloor(x))$ by definition of Inferable_{T}. Suppose to obtain a contradiction that $\neg\underline{\downarrow}(\lfloor p \rfloor(x))$. Then $\nvdash_{T} \underline{\downarrow}(\lfloor p \rfloor(x))$ by consistency of T.
2. Suppose $\underline{\downarrow}(\lfloor p \rfloor(x))$. Then $\vdash_{T} \underline{\downarrow}(\lfloor p \rfloor(x))$ by Adequacy of T for computation. It follows that $\text{Inferable}_{T}(\lceil \underline{\downarrow}(\lfloor p \rfloor(x)) \rceil) \rightarrow_1 \mathit{True}$.

But this contradicts $\neg\text{ComputationallyDecidable}[\text{HaltingProblem}]$ because $\text{Halt}(p, x) \Leftrightarrow \text{Inferable}_{T}(\lceil \underline{\downarrow}(\lfloor p \rfloor(x)) \rceil)$
Consequently, $\text{Consistent}_{T} \Rightarrow \neg\text{ComputationallyDecidable}[\text{InferenceProblem}_{T}]$

xxvii For example the following paradoxes prove every sentence:

- *Curry's Paradox* **[**Curry 1941**]:**
  $\text{Curry}_{v} \equiv \lfloor \text{Fix}(\text{Diagonalize}) \rfloor_{T}$
  where $\text{Diagonalize} \equiv (s \in \text{Sentences}) \rightarrow \lceil \lfloor s \rfloor_{T} \vdash_{T} \lfloor v \rfloor_{T} \rceil_{T}$
  1) $\text{Curry}_{v} \Leftrightarrow_{T} (\text{Curry}_{v} \vdash_{T} \lfloor v \rfloor_{T})$ and for any sentence v, it is possible to infer $\lfloor v \rfloor_{T}$ as follows:
  2) $\vdash_{T}(\text{Curry}_{v} \vdash_{T} \text{Curry}_{v})$ ⓘ *idempotency*
  3) $\vdash_{T}(\text{Curry}_{v} \vdash_{T}(\text{Curry}_{v} \vdash_{T} \lfloor v \rfloor_{T}))$
     ⓘ *substituting* 1) *into* 2)
  4) $\vdash_{T}(\text{Curry}_{v} \vdash_{T} \lfloor v \rfloor_{T})$ ⓘ *contraction*
  5) $\vdash_{T} \text{Curry}_{v}$ ⓘ *substituting* 1) *into* 4)
  6) $\vdash_{T} \lfloor v \rfloor_{T}$ ⓘ *chaining* 4) *and* 5)
- *Löb's Paradox* **[**Löb 1955**]:**
  $\text{Löb}_{v} \equiv \lfloor \text{Fix}(\text{Diagonalize}) \rfloor_{T}$
  where $\text{Diagonalize} \equiv (s \in \text{Sentences}) \rightarrow \lceil (\vdash_{T} \lfloor s \rfloor_{T}) \vdash_{T} \lfloor v \rfloor_{T} \rceil_{T}$
  1) $\text{Löb}_{v} \Leftrightarrow_{T} ((\vdash_{T} \text{Löb}_{v}) \vdash_{T} \lfloor v \rfloor_{T})$ and for any sentence v, it is possible to infer $\lfloor v \rfloor_{T}$ as follows:
  2) $\vdash_{T}((\vdash_{T} \text{Löb}_{v}) \vdash_{T} \text{Löb}_{v})$
     ⓘ *a proposition holds when it is inferred*
  3) $\vdash_{T}((\vdash_{T} \text{Löb}_{v}) \vdash_{T}((\vdash_{T} \text{Löb}_{v}) \vdash_{T} \lfloor v \rfloor_{T}))$
     ⓘ *substituting* 1) *into* 2)
  4) $\vdash_{T}((\vdash_{T} \text{Löb}_{v}) \vdash_{T} \lfloor v \rfloor_{T})$ ⓘ *contraction*
  5) $\vdash_{T} \text{Löb}_{v}$ ⓘ *substituting* 1) *into* 4)
  6) $\vdash_{T} \lfloor v \rfloor_{T}$ ⓘ *chaining* 4) *and* 5)