Interaction, Computability, and Church's Thesis

Peter Wegner, Brown University <pw@cs.brown.edu>
Dina Goldin, University of Massachusetts at Boston <dqg@cs.umb.edu>

Draft, May 25, 1999

Abstract: This article formalizes the claim that interactive finite computing agents are more expressive than Turing machines. The impact of models of interaction on Church's thesis and Godel's incompleteness result is explored. The evolution from algorithmic to interactive models of computation in computer architecture, software engineering, and AI is considered in a final section.

Contents

In computer science, important concepts usually come with a plethora of alternative characterizations.
Christos Papadimitriou [Pa]

1. Interaction Machines

Algorithms and Turing machines (TMs) have been the dominant model of computation during the first 50 years of computer science, playing a central role in establishing the discipline and providing a deep foundation for theoretical computer science. We claim that TMs are too weak to express interaction of object-oriented and distributed systems, and propose interaction machines (IMs) as a stronger model that better captures computational behavior for finite interactive computing agents. We show that the robust equivalence of TMs, the lambda calculus, and recursively enumerable sets in expressing the computable functions is paralleled by an equally robust equivalence of models and mechanisms of interactive computing. Moreover, changes in technology from mainframes and procedure-oriented programming to networks and object-oriented programming are naturally expressed by the extension of models of computation from algorithms to interaction.

Section 1 introduces interaction machines, an observation-based metric for interactive expressiveness, and expressiveness hierarchies for sequential and distributed interaction. Section 2 extends machines, sets,

algebras, and Church's thesis from algorithms to interaction. Section 3 presents non-well-founded set theory as a framework for interactive semantics, while section 4 extends induction to coinduction by removing the base case and replacing least by greatest fixed points. Section 5 examines metamathematical implications, including the extension of Church's thesis, the relation between mathematical paradigms of Brouwer and Hilbert and the computational paradigms of Church and Turing, and the interpretation of Godel incompleteness. Section 6 shows that interactive models provide a unifying framework for software engineering, AI, HCI, and other application-oriented subdisciplines of computer science.

1.1 Sequential Interaction Machines (SIMs)

Turing machines are finite computing agents that noninteractively transform input into output strings by sequences of state transitions. TM computations for a given input are history-independent and reproducible, since TMs always start in the same initial state and shut out the world during computation.

Turing machine: TMs are state-transition machines M = (S, T, s_0, F), with finite sets of states S and tape symbols T, a starting state s_0, and a state-transition relation F: SxT -> SxO. TMs transform finite input strings x∈ T* to outputs y = M(x) by a finite sequence of steps that read a tape symbol i, perform a state transition (s,i) -> (s',o), write a symbol o, and/or move the reading head one position right or left [Tu1].

TMs compute functions f:X->Y from integers to integers (strings to strings). The class of functions computable by TMs are called the computable functions.

SIMs are stream processing machines that model sequential interaction by I/O streams.

Sequential interaction machine: SIMs are state-transition machines M = (S, I, m) where S is an enumerable set of states, I is an enumerable set of dynamically bound inputs, and the transition mapping m: SxI -> SxO maps state-action pairs into new states and outputs.

Each computation step (s,i)->(s',o) of a SIM can be viewed as a complete TM computation, where i is a dynamically supplied input token (string), the output o may affect subsequent inputs, and s' is the next state of the SIM. Elements of both S and I are finite at any given time but their size is unbounded.The behavior of SIMs is expressed by *I/O streams:*

I/O streams have the form (i_1,o_1), (i_2,o_2), ... , where o_k is computed from i_k but precedes and can influence i_{k+1}. The dependence of i_{k+1} on o_k is called **input-output coupling**.

Input-output coupling violates the separation of domains and ranges, causing dynamic dependence of inputs on prior outputs that is characteristic of interactive question-answering, dialog, two-person games, and control processes. Transitions m from s_k to s_{k+1} I/O pairs are associated with I/O pairs (i_k, o_k).

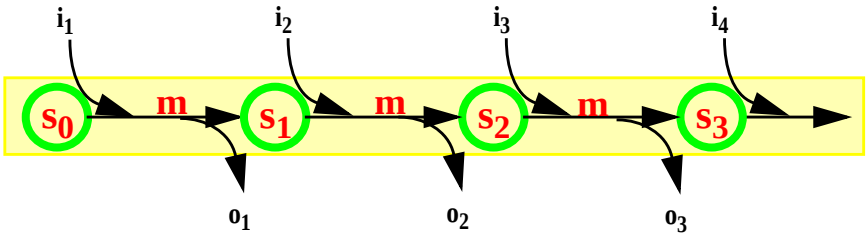


Figure 2: SIM with Input i∈ I, Output o∈ O, and mapping m

Persistent Turing machines (PTMs) are a canonical model for SIMs that minimally extends TMs [GW].

Persistent Turing Machine: A PTM is a multitape TM with a persistent work tape whose content is preserved between interactions.

The state of a PTM is its persistent worktape and may be enumerably infinite. Single interactions of a PTM correspond to TM computations.

Example: An answering machine is a PTM whose worktape contains a sequence of recorded messages and whose operations are "record message", "playback", and "erase". History dependent behavior of the answering machine is illustrated by the I/O stream segment "(record X, ok), (record Y, ok), (playback, XY)", where the third output "XY" depends on the first and second input. Both the content of the worktape and the length of input for recorded messages are unbounded, but we show [GW2] that "simple" PTMs

whose inputs are binary have the expressiveness of a PTM with unbounded input message.

1.2 Interactive Behavior and Expressiveness

Expressiveness of finite computing agents is defined in terms of a notion of observable behavior that generalizes the notion of algorithm behavior.

System behavior of a finite computing agent M is specified by its set of all possible interactions with all possible environments, and is denoted by B(M).

TM behavior is specified by a set of I/O pairs, while SIM behavior is specified by a set of I/O streams. SIMs have the same transformation mechanisms as TMs but have richer environment interaction that enables them to express richer behavior. Multi-agent interaction machines (MIMs), that interact with multiple autonomous interaction streams, have even richer interaction environments (section 1.3) that approximate to the interaction of physical objects in the real world.

Observable behavior of agents relative to a class of environments is generally more relevant in modeling interactive applications than system behavior for all possible environments.

Observable behavior of an agent (machine) M for an environment (observer) E is the set of all behaviors of M in E and is denoted by $B_E(M)$.

Observable behavior is a projection of system behavior to a specific context. The system behavior of M is defined as a union of behavior over all possible observation environments $B(M) = \cup E.\ B_E(M)$. The interpretation of "all possible" depends on the range of interactions considered legitimate. Turing machine interactions permit only I/O pair observations and exclude I/O streams, while SIM interactions permit I/O streams but exclude multiple I/O streams. When considering agents that simulate interaction with the real world, multi-stream environments with the interaction power of MIMs must be considered.

Given two finite agents M_1, M_2, the exclusive-or $B_E(M_1) \oplus B_E(M_2)$ is called the distinguishability set DS for M_1, M_2 in E. Members of DS are called *distinguishability certificate*s. Two machines are distinguishable if DS is nonempty and are equivalent if they are not distinguishable. Agents equivalent relative to a restricted environment E_1 may have distinct behavior for a more permissive environment E_2, just as for people. Greater problem-solving power of agents (or capability of people) becomes manifest only when agents are tested under demanding conditions that allow capabilities to be exercised.

Expressiveness: Machine M_1 is more expressive than machine M_2 in environment E if $B_E(M_1)$ is a strictly larger set of behaviors than $B_E(M_2)$.

Expressiveness of finite computing agents can be formally defined by observation equivalence relations that measure the ability of agents to make observational distinctions about their environment.

Observation equivalence: A class S of systems and a class E of environments determines an observation-equivalence relation EQ(S, E) such that systems M_1, $M_2 \in S$ are equivalent iff their behavior is indistinguishable for all $e \in E$. That is, $B_e(M_1) = B_e(M_2)$ for all $e \in E$. Environments e, $e' \in E$ are equivalent for systems S if they induce indistinguishable behavior $B_e(M) = B_{e'}(M)$ for all $M \in S$.

Machines induce an equivalence relation on an environment E such that machine M_1 is more expressive than M_2 if the equivalence relation $EQ(M_1, E)$ induced by M_1 on E is strictly finer than $EQ(M_2, E)$. Greater expressiveness of machines implies greater distinguishability among environments, which in turn means that more expressive machines can solve larger classes of computational problems.

Observation equivalence relations induced by finite automata were studied by Myhill and Nerode [RS]. They showed that a finite automaton induces an equivalence relation of finite index on its environment (set of tapes): finite automata can make only a finite number of distinctions about their environment. The Myhill/Nerode result can be generalized to show that TMs and IMs induce stronger equivalence relations and are therefore more expressive.

Lemma: The equivalence relation induced by a TM on its tape environment has an enumerable index.

Proof: A TM that defines a one-to-one function on an enumerable domain (for example, the successor function), defines an equivalence relation of enumerable index on its set of tapes.

Lemma: SIMs induce an equivalence relation with nonenumerable index on their environment.

Proof: SIM inputs are streams (models by non-well-founded sets) that distinguish among nonenumerable environments.

By extending the Myhill-Nerode results about finite automata beyond TMs to SIMs we show the surprising result that finite computing agents can make nonenumerable distinctions about their environments. This directly describes the greater expressiveness of interaction in terms of methods borrowed from the early history of automata theory.

finite automata induce an equivalence relation of finite index on their environment (of tapes)

TMs induce an equivalence relation of enumerable index on their (tape) environment

SIMs induce equivalence relations of nonenumerable index on their (external) environments

Interaction between an environment and an observer can be modeled as a producer/consumer relation, where E is a producer of behavior, S is a consumer, and the observed behavior (throughput) $B_E(S)$ is the smaller of produced and consumed behavior. When the environment with which a finite agent interacts is a TM, then behavior richer than a TM cannot be detected, just as the intelligence of a human agent cannot be detected if the agent is never given tasks that require intelligence.

Since the behavior $B_E(S)$ depends on E as well as S, we classify environments into classes that support progressively stronger forms of interaction.

Turing machine environments (agents noninteractively transform initial to final string)

SIM environments (agents respond to sequences of interactive inputs, proactively explore environment)

MIM environments (agents respond to autonomous (distributed) streams of interactive inputs)

the physical world W (interaction at least as demanding for agents as MIM environments)

Behavior of agents in a TM environment is expressed by input-output pairs, while behavior in a SIM environment is expressed by sequences of observations (I/O streams) that allow more finely-grained distinctions among behaviors than single observations. Nonequivalence of two SIMs can be detected by a finite distinguishability certificate (observation sequence) but equivalence cannot be finitely detected.

Since computational models aim to express properties of real-world environments, W is the most demanding of the environments considered here. Interaction machines that interact with the real world W, have the property that EQ(SIM, W) is a finer equivalence relation on the real world that EQ(TM, W). SIMs permit a finer granularity equivalence relation for real-world environments than TMs and MIMs permit a finer granularity relation than SIMs. Agents that model behavior of real-world environments W impose an expressiveness hierarchy such that TMs, SIMs, and MIMs are progressively more expressive finite agents, able to make progressively finer distinctions about the world and solve larger classes of problems.

Lemma: $B_W(TMs) \subset B_W(SIMs) \subset B_W(MIMs)$, *where* $\subset$ *is set inclusion for classes of behaviors*

Proof: The result $B_W(TMs) \subset B_W(SIMs)$ follows from the fact that TMs induce equivalence relations of enumerable index while SIMs induce equivalence relations of nonenumerable index. Evidence for $B_W(SIMs) \subset B_W(MIMs)$ is given in section 1.3.

For environments that are TMs, the equivalence relation induced by SIMs is no finer than that induced by TMs, while for SIM environments, MIMs are no more expressive than SIMs. A given level of expressiveness of a finite agent can be shown only if a testing environment has at least that expressiveness. To a TM observer, all computing agents including SIMs and MIMs appear to behave like a TM, while to a TM, all computing environments as well as the physical world appear to behave like TMs. Failure to observe that computers have greater expressive power than TMs was due in large measure to the fact that observation environments (testing environments) were restricted to be TMs.

$B_{TM}(TMs) = B_{TM}(SIMs) = B_{TM}(MIMs)$; $B_{TM}(TMs) = B_{SIM}(TMs) = B_{MIM}(TMs) = B_W(TMs)$

Once the idea of expressiveness as distinguishing power is introduced, many levels of expressiveness may be distinguished. Let SIM_k be the class of SIMs restricted to no more than k interactions. For all k>0, the distinctions that can be made by SIM_{k+1} in environment W are greater than those by SIM_k. SIMs determine an infinite expression hierarchy with TMs at the bottom (corresponding to behavior $B_W(SIM_1)$).

Lemma: *For all k>0, $B_W(SIM_k) \subset B_W(SIM_{k+1})$; and $B_W(TM) = B_W(SIM_1)$*

Proof: Consider an environment of SIMs with buffers of length k that buffer the output of their first k-1 interactions and disgorge their first output only on the kth interaction. For this environment, machines in the class SIM_k can distinguish machines with buffers of length k by distinguishability certificates of length k, but cannot distinguish among machines with buffers of length k+1. As a second example, consider an environment of machines M_k with binary inputs that print a 1 whenever the last k interactive inputs are 1 and a 0 otherwise. The shortest distinguishability certificate between M_{k+1} and M_k has length k+1. Note that SIM_k must have a memory of at least size log(k) to track distinguishability certificates of length k, so that the state for the classes SIM_k is not bounded.

SIMs model interactive question answering. The expressiveness hierarchy for SIMs shows that on-line questioning that makes use of follow-up questioning has greater expressiveness than off-line questioning. Ken Starr can obtain more information about questioned subjects (Clinton) by interactive than noninteractive questioning because later questions can use information not available to noninteractive questioners. Strategies that permit k+1 interactive questions can elicit more information (about Clinton) than k questions, assuming that Clinton has at least the expressive power of a SIM. This is formally proved by expressing interactive question-answering as interaction between two SIMs and using the result that SIMs can make finer distinctions about a SIM environment than TMs. The strategy used by the Senate of asking a list of 81 predetermined questions was predictably ineffective because it is an off-line strategy that corresponds to asking a single complex question with many parts.

1.3 Multi-Stream Interaction Machines (MIMs)

Multi-stream interaction machines (MIMs) are finite agents that interact with multiple autonomous streams: for example, distributed databases or ATM systems that provide services to multiple autonomous clients. MIM behavior is not expressible by SIMs [WG] in the sense that MIMs can make a richer class of observational distinctions (perform a larger class of tasks) than SIMs. This contrasts with the fact that multitape TMs are no more expressive than single-tape TMs. TMs express the behavior of a single agent, SIMs express the interaction of 2 agents, while MIMs express the interaction of n agents for n>2. Analysis of the greater expressiveness of 3-agent than 2-agent interaction provides a computational framework for showing that the 3-body problem in physics is not reducible to the 2-body problem [We3].

MIMs provide a precise definition of **distributed systems**: A system (computing agent) is distributed iff its interactions can be described by a MIM but not by a SIM.

This definition neatly defines distributed behavior for a single finite computing agent in terms of concurrency (nonserializability) of its interactive behavior. The notion that “distributed = interactively concurrent” is attractive because of its precision and simplicity. Its focus on concurrency of interaction as opposed to nonlocality of data may at first seem nonstandard, but in fact expresses distributedness in terms of a natural notion of nonlocality of agents. MIMs that cannot be expressed as SIMs express inherent nonlocality of the agents that interact with the MIM. Our claim that MIMs are more expressive that SIMs implies that distributed systems (defined as above) are more expressive than nondistributed systems.

MIM environments interact concurrently with finite agents. Interactive concurrency differs qualitatively from concurrent execution of actions and is a form of “second-order concurrency”. Concurrent systems with many parallel processes may be noninteractive (closed), interact with a single external stream, or interact with multiple streams [We1]. Distributed computing can be precisely defined as concurrent interaction: distributed systems have second-order concurrency while nondistributed systems interact with at most a single stream [WG].

Greater expressiveness of machine M_1 than M_2 for a multiple-stream environment E can in principle be observationally defined by the property that induced observation equivalence relation $EQ(M_1, E)$ is finer than $EQ(M_2, E)$, just as for SIMs. For example, a CEO M_1 who can draw finer distinctions about a company E than a CEO M_2 can make more effective decisions. However, equivalences induced by MIMs on

environments are harder to visualize and define than equivalences induced by SIMs, involving concurrent interaction with multiple streams that cannot be directly related to making environmental distinctions.

We present two kinds of arguments for the greater expressiveness of MIMs:

1. MIMs may have nondeterministic behavior not expressible by SIMs

MIMs model distributed systems that cannot be completely observed by any sequential observer (stream). Each stream of a MIM models an observer that views the MIM through a particular interface, but there is no observer that can observe the complete MIM. Observers of a MIM are like the proverbial observers of an elephant who can only observe the trunk or the tail but not the complete elephant.

Systems consisting of a SIM with an observer are *closed*, while MIMs with any primary observer are *open*, since they can be modified unpredictably by secondary observers (Figure 3). Modification of a MIM by secondary observers is perceived by a primary observer as nondeterministic behavior.

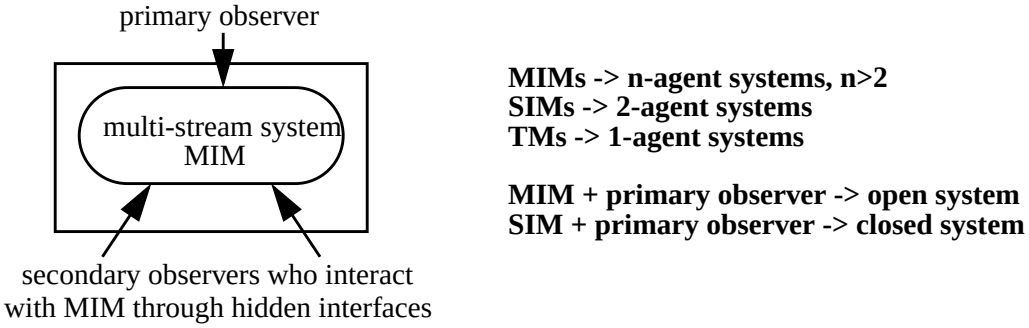


Figure 3: Multi-Stream Interaction of Primary and Secondary Observers

MIMs that are deterministic from the viewpoint of a metaobserver (God) who observes their complete behavior appear nondeterministic from the viewpoint of any single primary observer with limited sequential observation powers. This is a variant of Einstein's argument that "God does not play dice" which was the basis of his disagreement with traditional interpretations of quantum theory. Einstein's hidden variable model was proved inconsistent with observation [Per]. When Einstein's "hidden variable model" that express observation of SIMs are extended to "hidden interface models" that express observation of MIMs with unobservable (hidden) interfaces, the arguments of Bohr and Bell against Einstein do not apply. The hidden interface model potentially provides a stronger argument for the thesis that "God does not play dice" that could prove Einstein to be right. An experiment for deciding between Bohr's and Einstein's view of nondeterminism is proposed in [We3].

2. MIMs may have nonserializable behavior not expressible by SIMs

Nonserializability of autonomous streams provides a more direct argument than nondeterminism for the greater expressiveness of MIMs. Multiple autonomous streams cannot be serialized as a single stream because atomicity of input-output coupling of streams cannot be preserved under merging for tasks that require collaboration or coordination among streams. We illustrate nonserializability for delegation of subtasks - a particularly simple form of coordination.

If (iu, ou) is an input-output interaction of an agent A with a user U and (oe, ie) is an interaction by A with an expert E to whom A delegates a subtask in responding to U, then interactions occur in the temporal sequence (iu, oe, ie, ou). The user stream element (iu, ou) and expert stream interaction (oe, ie) cannot be treated as atomic and independent. There is no serial order of execution of the two interactions (iu, ou) and (oe, ie) that achieves the desired effect. This example shows that even simple forms of coordination like delegation cannot be expressed by SIMs, because of the breakdown of serializability.

Coordination violates serializability (atomicity) of input-output coupling of streams

Adding autonomous streams (observation channels) to an interactive computing agent increases its expressiveness, whereas adding noninteractive tapes to a TM simply increases the structural complexity of predetermined inputs, and does not affect expressive power. The restriction of MIM's to serializable behavior, just as that of SIMs to noninteractive behavior, trades tractability for problem-solving power.

MIMs are more expressive than SIMs, while multitape and single tape TMs are equally expressive

Serializing the effect of multiple threads (streams) restricts the problem-solving power of finite agents in important ways, since nonserializable behavior of computers and people is more important than is generally realized. Nonserializable behavior is a feature of collaboration that distinguishes high-level managers who handle interaction with multiple subordinates from assembly line workers who interact with a single stream [WG]. MIMs model important forms of higher level human behavior. The ability to characterize collaborative behavior by MIMs and to distinguish collaborative from sequential expressiveness may prove to be important in formalizing collaborative computing.

Collaboration, coordination, and management is modeled by MIMs but not by SIMs or PTMs

MIMs support the behavior of nonserializable transactions and true concurrency [Pr1], while SIMs support only serializable transactions and interleaving concurrency. We conjecture that the richer behavior of nonserializability and true concurrency can in principle be expressed by a form of observation equivalence that generalizes bisimulation. The abstraction from Turing machines to recursively enumerable sets is paralleled by the abstraction from SIMs to non-well-founded sets, which model the behavior of single streams. We conjecture (section 3.3) that MIMs can likewise be consistently specified by axioms of set theory whose precise form requires further research.

2. Extensions of Expressiveness

Algorithms express time-independent noninteractive transformation behavior of finite computing agents, but not the full range of interactive behavior of object-oriented, agent-oriented, or distributed computing systems. They are time-independent abstractions whose semantics is independent of both execution time and the time at which they are executed. Algorithmic complexity is deliberately defined independently of the speed of actual computers, depending only on number of executed instructions.

2.1 Interactive Extensions of Machines, Sets, and Algebras

The robust equivalent expressiveness of TMs, the lambda calculus, and recursively enumerable sets is paralleled by an equally robust equivalence of machine, set theoretic, and algebraic models for interactive computing. The expressiveness of interactive computing agents can be equivalently specified by machines [We1, We2], set theory [BM], and algebra [JR], just as for algorithms. IMs are the interactive analog of TMs, non-well-founded sets are the interactive analog of recursively enumerable sets, while coalgebras are the analog of algebraic models like the lambda calculus.

extending machine expressiveness: Turing machines -> interaction machines

extending set theoretic expressiveness: well-founded sets -> non-well-founded sets.

extending algebraic expressiveness: algebras -> coalgebras

Greater expressiveness can be captured by behavior of machines, specification power for sets, or reduction power for algebras. The underlying cause of greater expressiveness is the extension from construction-based inductive to observation-based coinductive methods of definition, reasoning, and modeling [WG]. *The equivalent expressiveness of TMs, algorithms, computable functions, and formal systems is due not to their maximal expressiveness but to their common reliance on induction for computation and reasoning.*

extending mathematical expressiveness: inductive models -> coinductive models

Induction determines enumerable collections of finite structures constructible from a base case (the integers), while coinduction determines nonenumerable collections of infinite structures (the real numbers). Induction is related to reducibility in the sense that inductively definable sets and systems can be constructed from (reduced to) sets of primitive elements. Coinductive sets and systems are not constructible from or reducible to primitive elements. Coinduction is related to abduction (inference from an agent's behavior to its inner structure) [We3] and to possible world semantics [Kr].

Once the idea that interactive computing is more expressive than algorithms is proposed, interactive extensions to both intuitive and formal notions of computation can be explored. Mathematical models that correspond to intuitive notions of interactive computing suggest equivalences between extended notions of intuitive and formal computation that are the basis for extensions of Church's thesis.

2.2 Interactive Extensions of the Church-Turing Thesis

The Church-Turing thesis has the form "the intuitive notion X corresponds to the formal notion Y". It equates the intuitive notion of effective computability of functions, shown by Turing to correspond to algorithmic computation, with the formal notion of TMs and the lambda calculus:

Church-Turing Thesis: The intuitive notion of effective computability for functions and algorithms is formally expressed by Turing machines (Turing) or the lambda calculus (Church).

X = computability by algorithms, Y = Turing machines, the lambda calculus

We accept Church's thesis, but distinguish it from the broader thesis that TMs completely express all forms of computability. Our position on Church's thesis is stated in [We1] and quoted in [PR]: "Though the thesis is valid in the narrow sense that TMs express the behavior of algorithms, the broader assertion that algorithms capture the intuitive notion of what computers compute is invalid". The assumption for functions from integers to integers that their input is completely given before the computation starts excludes interactive computation whose input may depend on events that occur during the process of computation.

In exploring the relation between intuition and formalism in computing we can ask the question "what is the formal notion Y for the intuitive notion X" or the very different question "what is the intuitive notion X captured by a formal notion Y". Church's thesis addresses the second question identifying an intuitive notion of computability as the semantic domain for formal models of TMs and the lambda calculus. However, premature commitment to a particular formal model for computing or any other discipline can lead to a narrow view of its subject matter by excluding phenomena that do not conform to the formal model.

We provide a definite answer to the question "what is the formal model that corresponds to intuitive notion of sequential interaction". The Church-Turing thesis can be generalized from algorithms to sequential interaction by extending both the intuitive notion X of computation and the associated formal notion Y.

Church-style thesis for sequential interaction: The intuitive notion of sequential interaction is formally modeled by non-well-founded sets.

X = computability by sequential (single-stream) interaction, Y = non-well-founded sets

Multi-stream interaction is shown to be more expressive than sequential interaction and it is conjectured that formal models for multi-stream interaction correspond to general models of coalgebras and coinduction. This suggests a further generalization of Church's thesis.

Church-style thesis for general interaction: The general intuitive notion of interactive computing is formally modeled by coinductive models (coalgebras).

X = general (multi-stream) interaction, Y = coinductive models of computation

These extensions of the Church-Turing thesis provide mathematical legitimacy for models of interaction by showing that intuitive notions of interaction have well-defined formal models (Figure 1).

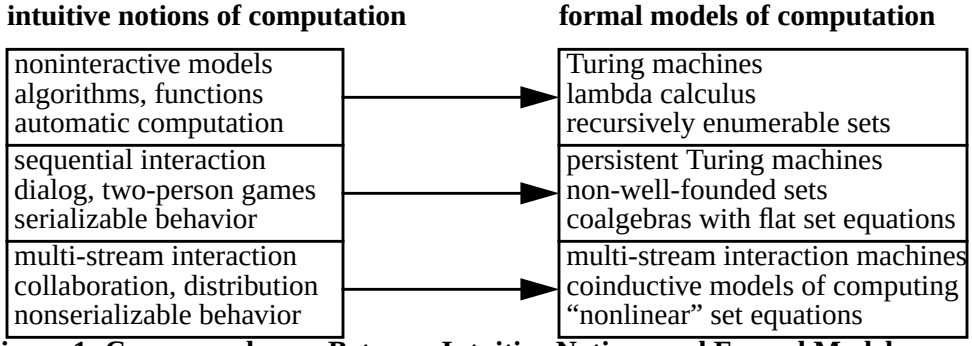


Figure 1: Correspondences Between Intuitive Notions and Formal Models

Reexamination of the intuitive notion of computability in the context of interactive software technology suggests a broader class of models of computation than those of Church and Turing, and provides the challenge of developing formal models that match new forms of intuition. This challenge has been met for sequential interaction by providing machine and set theory models that express this intuitive notion. For

multi-stream interaction there is as yet no definitive formal model, but some ideas for a framework for multi-stream interaction are presented in section 3. Section 5 examines the relation between intuitive notions and formal models of computing in greater detail.

3. Mathematical Models of Interaction

Set theory provides an abstract mathematical setting in terms of which semantic notions about computing can be formally expressed. Non-well-founded set theory extends inductive to coinductive methods of definition that determine a larger class of sets and allow reasoning and modeling of behavior for larger classes of computing systems and computational problems. Coinduction supports the definition of nonenumerable classes of infinite (non-well-founded) structures and supports consistent methods of circular reasoning as well as a notion of equivalence related to bisimulation. Non-well-founded set theory expresses the semantics of interaction for single streams, while general coinductive reasoning provides a semantic framework for multi-stream interaction.

3.1 Non-Well-Founded Set Theory

Sets in traditional set theory are inductively constructible from atomic elements by union, intersection, complement, cross-product and powerset constructors. Powersets define sets that contain sets as elements, but any constructively definable set is well-founded in the sense that the process of defining sets in terms of other sets is finitely grounded.

Well-foundedness of sets is expressed by well-foundedness of relations. A binary relation R on a set S is *non-well-founded* if there is an infinite sequence $b_i \in S$ for i = 0,1,2,..., such that $b_{i+1} R b_i$ for all i, and is *well-founded* otherwise. Well-founded sets are defined in terms of well-foundedness of the set membership relation $\in$. A set is well-founded if the set membership relation over its structure is well-founded and is non-well-founded otherwise: w*ell-founded sets* have only finite set-membership chains, while *non-well-founded sets* may have infinite set-membership chains.

Zermelo-Frankel set theory provides an axiomatic specification ZF of sets that can be supplemented by the *foundation axiom* to obtain the axiom system ZFC.

foundation axiom (FA): all sets are well-founded (ZF + FA = ZFC)

The well-founded sets definable by ZFC are precisely the inductively constructible (recursively enumerable) sets. A set has a finite set membership chain iff it can be inductively constructed from primitive elements by a finite number of construction operations. However, ZFC excludes non-well-founded sets, which can be defined as the solutions of systems of flat set equations.

flat set equations: A system of flat set equations has the form (S, A, m), where S is a set of variables (states), A is a set of constants (actions, observations) and m: S -> P(A$\cup$S) is a mapping that specifies for each $s \in S$ a set of constants in A and variables in S.

Extending set theory by admitting solutions of set equations is analogous to extending the integers to rationals by admitting solutions to equations with integer coefficients. **Non-well-founded set theory** augments ZF with the *anti-foundation axiom (AFA)* to obtain ZFA. ZFA is consistent if ZF is consistent [BM].

Anti-Foundation Axiom: Every flat system of equations has a unique solution (ZF + AFA = ZFA).

Equations of the form m: S = P(AxS) whose right hand sides are sets of ordered pairs can be expressed as flat equations by the reduction (a,b) -> {a, {a,b}} [BM]. Equations S = P(AxS) are an important subclass of flat equations that model the observable behavior of state transition systems. Ordered pairs (a,s) model the observation semantics of systems for which an observation a causes a transition to state s.

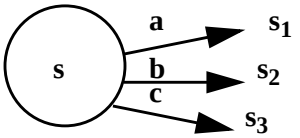
Definition: A *stream* over A is an ordered pair s = (a, t), where $a \in A$ and t is another stream. The set S of all streams over A is the maximal solution (maximal fixed point) of the equation S = AxS.

Example: The solution of the set equation s = {(0,s), (1,s)} specifies an infinite set of binary streams, illustrating that non-well-founded sets may be nonenumerable.

Solutions of flat set equations are well-founded if there is no circularity in their definition as in s = {a,t}, t = {a,b} and are non-well-founded otherwise. The solution of s = {(0,s), (1,s)} does not satisfy FA

and cannot be inductively generated. It is not a member of ZFC, illustrating that the class ZFA is strictly larger than ZFC (models a larger class of systems).

Non-well-founded sets can model **labeled transition systems** (LTSs). An LTS with n states can be specified by a system of n simultaneous set equations (S, A, m), where m:S -> P(AxS) maps states into subsets of action-state pairs. For example, a state with outgoing edges labeled a to s1, b to s2, and c to s3 is represented by the equation s = {(a,s1), (b,s2), (c,s3)} (Figure 4).



Equation for state s of an LTS
$s = \{(a, s_1), (b, s_2), (c, s_3)\}$

Figure 4: Set equation for State with Three Outgoing Edges

The labels (actions) of an LTS should be interpreted as input-output pairs (i_k, o_k) of an I/O stream. This is consistent with Figure 2 for SIMs, which shows an I/O pair labeling each state transition edge. The unfolded behavior of an LTS can determine a nonenumerable set of possible behaviors that are solutions of the set equations representing the LTS. Coinductively specified finite agents may have nonenumerable interactive behaviors, while noninteractive finite agents (TMs) have enumerable behaviors.

3.2 Coalgebras

Coalgebras are a model for interactive computing that parallels algebras as a modeling framework for algorithms. Algebras specify incremental expression evaluation whose goal is to compute a value, while coalgebras specify incremental system observation whose goal is to determine system behavior. Operations of algebras are computation steps, while operations of coalgebras are interaction (observation) steps. Equivalence classes of expressions that have the same value are enumerable, while equivalence classes of systems (programs) that have the same behavior are not.

Algebras are structures A = (S, m: F(S) -> S), where S is the carrier set and F is a functor that determines the signature of A. We usually interpret S as a set of values and m: F(S) -> S is a value-preserving homomorphic mapping m from a syntactically specified set of expressions into a value set. Algebras determine reduction processes (mappings) from an inductively defined *initial algebra* of syntactically specified expressions to a quotient algebra that determines the value set.

Coalgebras are structures CA = (S, m: S -> Γ(S)), where S is a carrier set and Γ is a functor that determines the signature of CA. We usually interpret S as a set of states of an observed system and m: S -> Γ(S) as a behavior-preserving homomorphism of observed systems with an unknown state into unfolded set of behaviors (observation sequences).

Coalgebras whose mappings are one-to-one are called *final coalgebras*. States of final coalgebras represent unfolded system behaviors and, for mappings m that are expressible as flat equations, are specified by non-well-founded sets. Final coalgebras are canonical behavior specifications for equivalence classes of systems with the same behavior that are a coalgebraic analog of canonical value specifications (normal forms) for algebraic values. Initial algebras are an inductively specified set of expressions, while final coalgebras are coinductively specified sets of behaviors.

Coalgebraic homomorphisms that map systems into their behaviors are stronger than algebraic homomorphisms that map expressions into values because equivalence classes of systems with the same behavior (for example, programs that compute the same function) are nonenumerable, while classes of expressions with the same value are enumerable. The greater richness of non-well-founded over well-founded sets translates into greater computation power of coalgebras over algebras.

Coalgebras that specify non-well-founded sets and model sequential interaction have mappings of the form m: S -> P(A∪S), corresponding to the AFA. It is shown in [BM] that mappings such as m: S -> P(AxS) of LTSs or mappings m: S -> OxP(AxS) of automata are expressible as flat systems of equations.

3.3 Beyond Non-Well-Founded Sets

The extension to non-well-founded sets is specified axiomatically by postulating that every "linear" (flat) set equation of a certain form has a solution. Though there is no corresponding axiomatic extension for single-stream to multi-stream models, we conjecture that such an extension could be specified by extending the requirement that set equations have a solution from linear to nonlinear equations, where the number of streams corresponds to the degree of the equation, and each of the n solutions corresponds to the behavior of a single stream (perceptions of a single observer). The coinductive dual of nonlinear equations remains to be developed, but the notion that coinductive nonlinear equations of degree n have n solutions which can be interpreted as streams of a MIM is an attractive one.

Equations of an LTS of the form $s = \{(a,s_1), (b,s_2), (c,s_3)\}$ can be written in the notation of process algebra as $s = a.s_1 + b.s_2 + c.s_3$ for a suitable interpretation of . and +. In this second form we can represent an LTS by the equation S = A.S. This can in principle be rewritten as (A-I).S = 0, so that the fixed point solution (non-well-founded set) S can be viewed as an eigenvector of the matrix equation.

MIMs with n autonomous streams can be represented as nonlinear systems of the form $m_1*m_2*...*m_n$, where m_i is the mapping function for stream i and * is a yet to be defined product operation whose properties reflect interaction among autonomous streams. We expect * to be associative and symmetric, reflecting that the behavior of MIMs should not depend on stream order.

Consistent extensions of set theory that admit larger sets than the non-well-founded sets are clearly possible [WG]. In principle, any finite agent can be modeled by a consistent extension of set theory that expresses the class of behaviors computable by the agent. Thus there is a consistent extension of set theory that expresses the behavior of a MIM. The conjecture of the previous paragraph makes a stronger claim of decomposability of behavior into "factors" associated with individual streams of the MIM. Put another way the conjecture asserts that behavior of a multi-stream agent may be compositionally defined in terms of behaviors of single streams. This conjecture corresponds to strengthening the model theoretic principle "existence implies consistency" to the principle "consistency implies existence", which is discussed at length in section 5.3. The formalization of multi-stream interaction is a subject of future research.

Coalgebraic mappings of the form S -> Γ(S) are "Markovian" in the sense that the new state depends only on the previous state. Non-Markovian mappings of the form Γ(S) -> Γ'(S) can model systems whose next state depends on several previous states or on several streams. The remarks in this subsection are speculative, but may provide pointers for a set theory of multi-stream models of interaction.

4. From Induction to Coinduction

Induction is a process of definition and reasoning that determines enumerable sets of finite structures reachable from an initial base case. The integers are the prototypical inductively definable set: they are an enumerable set generated from the initial element 0 by the successor function. Coinduction is a stronger form of definition and reasoning for nonenumerable sets of infinite structures. We examine the mechanisms of induction and suggest that coinduction is derivable from induction by eliminating the requirement of a base case and replacing least by greatest fixed points. Eliminating the base case extends closed to open systems while replacing least by greatest fixed points extends construction to observation paradigms.

The assertion "God created the integers and all the rest is the work of man" implies that induction is a sufficient and complete reasoning principle for understanding the world. Coinduction requires stronger creation principles (ontology) and extends mathematical modeling power from inductively definable non-interactive construction processes to observation processes for interactive finite agents.

4.1 The Inductive Modeling Paradigm

Induction determines a construction paradigm for definition, reasoning, and modeling characterized by:

(1) An *initiality condition* that determines base elements of inductively generated sets

(2) An *iteration condition* that allows new elements to be constructed (derived) from initial elements

(3) A *minimality condition* that only elements so constructed are definable

inductive definition: initiality (base) condition, constructive iteration condition, minimality condition
construction paradigm: generate (construct) structures inductively from base elements

Induction is the basis for defining sets of strings, languages, formal systems, and computable functions. The set A* of all strings over an alphabet A has the following base, iteration and minimality condition:

Strings: (1) the empty string e∈ A*; (2) if x∈ A* then ax∈ A*; (3) A* contains no other elements.

Languages L(G) are defined by grammars G over terminal symbols T and nonterminal symbols N as sets of strings inductively generated from an initial nonterminal S by generating rules (productions) P.

A **language** L(G) over T is the set L(G) = {x | S => x and x∈ T*}, where G = (N, T, S, P) is a grammar with nonterminals N, terminals T, initial nonterminal symbol S and productions P.

(1) symbol S; (2) productions P; (3) only inductively generated strings in T* are in L(G)

Theorems of a formal system are inductively derived from axioms by rules of inference.

A **formal system** FS = (AX, TH, RI) is an inductive specification of a set TH of theorems:

(1) axioms AX; (2) rules of inference RI; (3) only inductively generated formulae are theorems.

Computable functions f: X -> Y are inductive at two distinct levels. They map an inductively defined domain X to a range Y by inductively defined computing processes [Tu1]. TMs transform inductively-defined strings by inductively defined state-transition steps. This condition is sufficient as well as necessary: computable functions can be defined as inductive computations over inductively defined domains.

Inductively defined domain (statics): The domain X is inductively defined.

Inductively defined computation (dynamics): The process of computation is inductively defined.

Formal systems likewise make use of two levels of inductive definition. Well-formed formulae and axioms are defined by static induction, while processes of proof are defined by dynamic induction. The set of provable theorems is defined by inductively derivable formulae from inductively defined axioms.

The robust equivalence of TMs, the lambda calculus, RE sets, and functions from integers to integers is due to their reliance on induction as the principle of definition and reasoning. However, in spite of its robustness, induction is not the strongest possible definition and reasoning principle. Coinduction is a stronger principle that makes use of circular reasoning to handle partially observable environments, allowing finite computing agents to perform tasks with nonenumerable interactive behaviors.

4.2 Coinduction and Greatest Fixed Points

Coinduction determines a *deconstruction paradigm* that deconstructs composite structures into progressively more primitive ones. Coinduction models processes of *observation*. Observation is a deconstruction process that reveals progressively more structure about observed entities. Processes of observation make no initiality assumptions and can continue to reveal new knowledge about the observed objects indefinitely: hidden information is progressively approximated by processes that do not terminate.

Coinduction can be derived from induction by relaxing requirements of induction as follows:

(1) Eliminate the initiality condition (base case)

(2) Reverse the direction of iteration: construction -> deconstruction, observation

(3) Replace minimal by maximal fixed points

coinductive definition: deconstructive iteration condition, maximality condition.

observation paradigm: observe already existing constructed elements.

By expressing induction and coinduction in terms of more primitive concepts, we can separately consider the effect of eliminating initiality, reversing iteration, and replacing minimality by maximality in definition and reasoning processes. Eliminating initiality, combined with reversal of iteration, corresponds to elimination of finite coinductive termination (finite finality). It removes the closed-system requirement of complete environment specification before the computation starts, thereby modeling open systems.

Non-well-founded sets, defined as solutions to sets of flat equations (section 3), show that coinduction can define nonenumerable sets. Stream inputs for IMs are coinductively defined (section 3). Sets that are solutions of recursive stream equations do not exist in traditional set theory, where stream equations have the trivial empty set as their solution. The minimal fixed point of the equation S = AxS is the empty set,

while the maximal fixed point is the set of all streams over S.

Minimality, modeled by least fixed points, is a property of constructive processes of computation and constructive mathematics, while maximality, modeled by greatest fixed points, is a property of empirical observation paradigms for describing observed behavior in an already constructed (already existing) world. Minimality of behavior is associated with maximality of constraints on behavior, while maximality of behavior is associated with minimality of constraints. Maximal fixed points (minimal constraints) provide a mathematical framework for the empirical paradigm that any behavior (possible world) consistent with observation is admissible. Specifications that admit any possible world consistent with a specification are maximal fixed points (minimal constraints on behavior). The distinction between possible-world semantics of Kripke [Kr] and traditional model theory is precisely that of maximality versus minimality. Distinctions between restrictive and permissive social organization, such as those between totalitarian and democratic societies, are modeled by minimality versus maximality. Minimality models centrally controlled structures while maximality admits distributed control.

totalitarianism embodies the minimality principle: **everything is forbidden that is not allowed.**

democracy supports the maximality principle: **everything is allowed that is not forbidden.**

Non-well-founded set theory extends traditional set theory with sets that are solutions to recursive set equations. It provides a framework for formalizing sets with a coinductive (non-well-founded) structure, and correspondingly extends the class of models that sets can formalize to include interactive models of computation. Coinduction extends the definition and reasoning ability of finite computing agents so they can model nonenumerable sets defined by streams.

Algorithmic denotational semantics is specified by lattice-structured approximations to least fixed points that determine computable functions [Sc], while traditional operational semantics expresses the state-transition structure of algorithm execution steps. Non-well-founded sets express richer denotations than well-founded sets, specifying maximal fixed points. Non-well-founded set theory provides a denotational semantics for streams, while coalgebras provide a framework for operational semantics of interaction machines that are transducers of streams [WG].

5. Metamathematics of Coinduction

In this section, we examine the relation between intuitive and formal notions of computing of Church's thesis, consider Godel-style incompleteness theorems for coinductive models of computation, and reexamine notions of constructive, formalist, and realist mathematics for coinductive modeling.

Church and Turing's work on computability in the 1930s was strongly influenced by the intense foundational debates between intuitionists and formalists in the 1920s [De]. Church's thesis, that relates "intuitive" to "formal" notions of computing, can be viewed as a computational counterpart of Hilbert's thesis that intuitive notions of mathematics can be expressed by formal notions of logic. Both are completeness conjectures that claim complete expressibility of intuitive by formal notions, and both are true for inductive models but false for coinductive models.

Coinduction models stronger behavior of finite agents than inductive intuitionist or formalist models because of an ontological commitment to the existence of a strictly larger class of mathematical objects (the non-well-founded sets). It provides stronger notions of formalism and intuition than inductive models. Thus the interactive Turing test [We3], which permits machines that think to be interaction machines, can model both stronger forms of intentionality that satisfy Searle and stronger forms of extensionality that satisfy Penrose. Coinductive models suggest new interpretations of both Brouwer's "intuitionist" belief that mathematical reasoning is based on inner intuitions, and of Hilbert's "formalist" thesis.

5.1 From Formal Models to Intuitive Notions

Theses that relate intuitive to formal models of computing, like Church's thesis, are motivated either by the desire to formalize a given intuitive notion or by the goal of providing intuition for a given formal concept. Church's thesis has the second motivation, providing intuitions for the robust formal concept of computability expressible by Turing machines, the lambda calculus, or partial recursive functions. The

equivalent expressiveness of formalisms provides strong evidence for the existence of a robust intuitive notion that Church called “effective computability”.

Church-Turing thesis: Formal effective computability by the lambda calculus (Church) or TMs (Turing) expresses the intuitive notion of effective computability of functions (over positive integers).

In the early years of computing the intuitive notion of computing was identified with algorithms, and the focus of research on the complexity, performance, and design of algorithms provided a solid foundation for computer science. As technology became increasingly interactive, software engineering and other interactive subfields of computer science experiences a crisis in part because algorithmic foundations could not support interactive application technology. But the intuitive notion of computing continued to be formalized by TMs because no formal model beyond that of TMs or well-founded set theory was available. Non-well-founded set theory and SIMs provide well-defined mathematical and machine models that go beyond algorithms, allowing Church’s thesis to be extended.

We agree that effectively computable functions from integers to integers are modeled by TMs, but claim that the robustness of Church-Turing models is due to their common inductive basis rather than their ability to completely express all forms of computation. Coinductive models uniformly extend the expressiveness of machines, algebras, and set theory. Coinduction provides a more expressive mental tool for definition, reasoning, and modeling that shows Turing machines to be weak expressive models limited by induction. When the intuitive notion of effective computation is broadened to include interaction, formal computability must be correspondingly broadened to streams specified by non-well-founded sets.

Thesis for sequential interaction: Formal specifiability by non-well-founded sets, SIMs, or PTMs corresponds to the intuitive notion of effective computability for sequential (single-stream) interaction.

Both Church and Turing formalized computability in terms of functions f: X -> Y from integers to integers. Expressing domains as inductively definable integers requires complete specification of arguments before the computation starts and separability of domains and ranges: conditions assumed by Church and Turing but violated by I/O streams (section 2). Stream mappings f: stream -> stream are certainly not functions from integers to integers: whether they are classified as noncomputable functions (over noninductive domains) or computable nonfunctions (because mappings over noninductive domains are not considered functions) is a matter of definition. The question of whether computable mappings over noninductive domains are functions has not, to the authors’ knowledge, been formally settled.

The Church-Turing thesis is a conjecture about the restricted notion of algorithmic computation rather than about the broader intuitive notion of computing. The thesis for sequential interaction is likewise a conjecture about a restricted notion of computing that non-well-founded sets are not the most expressive class of consistent set theoretic models and that more powerful consistent models are axiomatically definable. More expressive classes of computing mechanisms may be definable by replacing the AFA by a consistent axiom that specifies a larger class of sets. We further conjecture that there is no well-defined maximal class of axiomatically definable consistent sets, and that no well-defined formal notion of maximal effective computability exists. This suggests that any formal notion of effective computability is relative rather than absolute, definable only relative to assumptions about the form of interaction.

5.2 Reinterpreting Godel Incompleteness

Church’s thesis can be viewed as a completeness conjecture that TMs completely express the intuitive semantic notion of “effective computability”. Hilbert’s thesis that logic can completely express intuitive notions of mathematics is a comparable completeness conjecture. Godel’s incompleteness theorem showed the impossibility of reducing mathematics to first-order logic [Fe] and by implication of reducing the behavior of finite computing agents to logic [We3]. Godel proved his theorem by showing that arithmetic over the integers could not be expressed by an enumerable number of formulae, using diagonalization to prove nonenumerability. We believe that incompleteness for interaction machines can be proved by building on the fact that the interactive behaviors of interactive agents cannot be enumerated.

First-order logic can model only enumerable semantic domains because its number of theorems is enumerable, which is easy to prove. Proving that a particular semantic domain is nonenumerable can be hard,

but diagonalization is a useful tool that was used by both Cantor to prove nonenumerability of the reals and Godel to show nonenumerability of the true assertions about integer arithmetic. Godel's incompleteness result is a simple corollary of the "folk theorem" that inductively specified enumerable classes of theorems cannot completely model nonenumerable classes of objects or situations [We3].

First-order logic can model only inductively (recursively) enumerable semantic domains

To prove Godel incompleteness of any domain, show it is not enumerable

Diagonalization is a tool for showing that specific domains are not enumerable

Whereas mathematicians immediately realized the impact of Godel incompleteness on Hilbert's program, computer scientists have been slow to accept the implications of Godel's result. TMs are an expressively weak though formally strong notion of effective computability that parallel the expressively weak but intuitively strong notion of proof of intuitionist models of mathematics. Church's thesis is the computational analog of restricting mathematics to intuitionist methods of definition and reasoning. Though Church's thesis was proposed several years after Godel's incompleteness theorem, it is "pre-Godelian" in its claim that computing can be completely expressed by an inductive (constructive) model of computation

IMs have nonenumerable possible behaviors corresponding to non-well-founded sets of streams, while TMs have enumerable possible behaviors corresponding to sets of strings (functions over domains of integers). Incompleteness of sequential interaction can be directly proved by showing that non-well-founded sets cannot be expressed as well-founded sets. Godel's incompleteness result implies not only that inductive mathematics is too weak to express arithmetic over the integers, but also that interactive models are incomplete. The incompleteness of interactive models is easier to prove than incompleteness of arithmetic because interactive models are more strongly nonenumerable than recursively enumerable sets, being uncountable rather than merely not recursively enumerable.

Linear logic [Gi] provides a bisimulation-based semantics of interaction for two-person games [Ab] more fine-grained than that of computable functions. It is described in [Ab] as an "intensional" semantics of interaction that "interpolates between denotational and operation semantics as traditionally conceived". But in fact it extrapolates beyond traditional semantics by building a "second-order" operational semantics on top of a denotational semantics for functions. We conjecture that linear logic is complete for sequential interaction in the sense that it can express SIM behavior and non-well-founded set theory. However, exploring connections between linear logic and non-well-founded sets is beyond the scope of this paper.

We further conjecture that multi-stream behavior of MIMs cannot be expressed by non-well-founded sets or linear logic and that MIM behavior can be expressed by consistent coinductive axiomatic specification. This suggests the following extension of the Church-Turing thesis for general interactive behavior.

Coinductive Church-Turing thesis: Coinductively specifiable behaviors expressible by axiomatic set theory correspond to the intuitive notion of computations expressible by finite computing agents.

This completeness conjecture asserts that coinduction can completely express interaction. Each Church-Style theses can be viewed as a completeness result about the characterization of an intuitive by a formal notion of computation, while the hierarchy of progressively stronger theses determine incompleteness results of the weaker thesis as an expression of formal or intuitive semantics of the stronger thesis.

Godel credits his mathematical successes to his Platonic belief in the independent reality of mathematical objects, arrived at early in his university years [Fe]. His proof that arithmetic for integers could not be formalized was motivated by his belief in "philosophical realism". But Godel concealed his Platonist (realist) beliefs, using Platonist principles as a basis for research rather than as a topic for analysis.

5.3 From Coinductive to Realist Ontology

Non-well-founded sets can model larger classes of objects, situations, and computational problems than well-founded sets because axioms of ZFA admit existence of a larger class of sets than axioms of ZFC. The existential (ontological) strength of mathematical paradigms determines the expressiveness of their models. Questions of mathematical existence are central to Russell's attempted reduction of mathematics to logic, Brouwer's intuitionism, Hilbert's formalism, and Godel's incompleteness result [De].

Logicians were reluctant to accept coinductive models because they failed to distinguish between

inconsistent and consistent forms of circular reasoning, following Russell in overreacting to the paradoxes of set theory [BM]. An additional reason for excluding coinductive thinking from mainstream mathematics was the restrictive influence on logic of constructive mathematics exemplified by Brouwer's intuitionism.

Hilbert in principle accepted the Platonic position that consistency (of a formalism) implies existence (of a model). But in practice he restricted formalism to inductive definition and reasoning, excluding consistent as well as inconsistent forms of circular reasoning. We refer to Hilbert's formalism as "inductive" formalism because he rejected consistent coinductive circular reasoning.

When Godel proved incompleteness of arithmetic by showing, through diagonalization, that true assertions of arithmetic cannot be inductively defined, the mainstream mathematical community accepted incompleteness as an absolute impossibility result for formalist mathematics rather than a relative result about the weakness of inductive reasoning. The alternative of adopting a stronger notion of formalism based on the acceptance of all forms of consistent reasoning was rejected.

Finsler's prescient work on set theory in the 1920s [Fi] showed the consistency of circular reasoning and anticipated Godel's incompleteness result, but was largely ignored because it did not conform to the mainstream paradigm of formalist mathematics. Finsler, influenced by Cantor's model of the real numbers, took to its logical conclusion the viewpoint that concepts exist independently of formalisms in which they are expressed. He went beyond Hilbert's formalism in applying the principle "consistency implies existence", accepting the existence of consistent sets of concepts independently of whether they are formalized. The discovery sixty years after the foundational discussions of the 1920s that non-well-founded set theory models interactive computing validates the view that conceptually consistent sets exist independently of their formalizability or constructibility. We call this viewpoint a *realist ontology* of mathematics.

Realism traditionally refers to models that accord an existence to modeled objects independently of whether they are perceived [EB]. By analogy, mathematical realism, as we define it, accords objects an existence independently of whether they are formalized. Realism in mathematics and empiricism in physics and computing both accord independent existence to objects being modeled, providing a foundation for empirical computer science and a basis for an interdisciplinary methodology of empiricism [We3].

Intuitionism, inductive formalism, and realism can be classified by their degree of commitment to the existence of mathematical objects, embodying progressively stronger forms of ontological commitment that can model progressively larger classes of applications.

intuitionism (Brouwer): existence requires (inductive) constructibility (minimal ontology)

inductive formalism (Hilbert): consistency of (inductive) formal systems implies existence of models

realism (Cantor, Finsler): consistency (of a specification) implies existence (maximal ontology)

Coinductive models of finite agents are realist in the sense that they can model both nonenumerable real-number domains and physical domains like the real world. They relate mathematical and physical meanings of the term "real" because coinduction models both mathematical nonenumerability and physical interaction. They determine an ontological paradigm shift from constructive to realist models. Constructive inductively defined ontologies specified by least fixed points are weaker than realist coinductively defined ontologies specified by greatest fixed points. Maximal fixed points admit interactive and time-dependent behavior for finite agents: they provide a mathematical foundation for realist ontology.

Hilbert's inconsistency in claiming to accept the realist principle "consistency implies existence" (C->E), but limiting its application to inductive formalism is a primary cause of Godel incompleteness. Feferman [Fe] does not adequately explain Godel's reasons for his surprising opposition to (C->E). Our analysis suggests that it may stem from the fact that C->E is incompatible with Godel's belief in objective mathematics and his acceptance of the inductive formalism of his teacher Hilbert. Godel's puzzling removal of an incisive discussion of C -> E between the initial and published version of his thesis [Fe], may well be due to his failure to resolve perceived contradictions between the principle of C->E and Hilbert formalism.

Claim: C->E, inductive formalism, and objective mathematics cannot be simultaneously true.

C->E, inductive models, and objective mathematics are incompatible starting points for a mathematical *Weltanschauung*. Hilbert, Godel, and Cantor (Finsler) each accept two and reject one of these principles.

Hilbert: C->E + inductive formalism; inconsistency with objective mathematics was proved by Godel

Godel: objective mathematics + inductive formalism; explains Godel's rejection of C->E
Cantor, Finsler: C->E + objective mathematics; implies rejection of inductive formalism

Hilbert's belief in C->E and inductive formalism was shown by Godel to be incompatible with objective mathematics, while Godel's belief in objective mathematics and inductive formalism caused him to reject C->E. The realist, coinductive paradigm of mathematics corresponds to Cantor and Finsler's belief in C->E and objective mathematics, which require rejection of inductive formalism.

Godel's incompleteness result was due to acceptance of the principle that inductively defined objects have a mathematical existence while coinductively defined objects do not. Had Godel instead accepted C->E and recognized along with Finsler that this implied acceptance of coinductive reasoning, his incompleteness result for inductive reasoning would have become a completeness result for coinductive reasoning and the evolution of logic might have been very different.

6. Interactive Software Technology

The evolution of computing in its first 50 years may, as a first approximation, be described as a transition from algorithmic to interactive computation. SIMs provide a domain-independent model of sequential interaction that spans architecture, software engineering, and AI, while MIMs provide a corresponding domain-independent model for coordination, collaboration, and distributed computation.

1950s: machine language, assemblers, hardware-defined action sequences
1960s: procedure-oriented languages, compilers, programmer-defined action sequences
1970s: structured programming, composition of action sequences, algorithm composition
1980s: object-based languages, personal computers, sequential interaction architecture
1990s: networks, coordination and collaboration, distributed interaction architecture

The 1950s through the 1970s were concerned with the development and refinement of algorithm technology for mainframes, sequential interaction became the dominant technology of the 1980s, while distributed interaction became the dominant technology in the 1990s. Whereas the shift from machine to procedure-oriented languages involves merely a change in the granularity of actions, the shift from procedures to objects is more fundamental, involving a qualitative extension from algorithmic to interactive finite computing agents. The extension from sequential to distributed interaction requires a further fundamental paradigm shift in models of computation. SIMs express the shift from algorithms to sequential interaction architecture, while MIMs express the further shift to distributed interaction.

Figure 5 illustrates the extension from algorithms to interaction along a number of dimensions. Each algorithmic concept in the left-hand column is paralleled by a more expressive interactive concept in the right-hand column. Moreover, each right-hand concept has both a single-agent (sequential) and a multi-agent (distributed) form whose expressiveness is specified by SIMs and MIMs.

Algorithmic Concepts	**Interactive Concepts**
input-output transformation	services over time (QoS)
procedure-oriented programming	object-oriented programming
structured programming	structured object-oriented prog.
compositional behavior	emergent behavior
programming in the small	programming in the large
logic and search in AI	agent-oriented (distributed) AI
closed systems	open systems
algorithmic computer science	empirical computer science

Figure 5: Parallel Extensions from Algorithms to Interaction

The transition from input-output transformations to interactive services over time arises in many different contexts. Services over time cannot be reduced to or expressed by algorithms or TMs. Algorithms are time-independent (instantaneous) episodes in the life-cycle of an interactive system. The one-dimensional

quantitative performance metric of algorithmic complexity becomes the multidimensional qualitative performance metric of quality of service (QoS), which is an increasingly central focus for research in the database and human-computer interaction communities.

Objects provide richer services to clients than algorithmically specified procedures. Procedures specify sales contracts, guaranteeing an output for every input, while objects specify marriage contracts, describing ongoing contracts for services over time. An object's contract with its clients specifies its behavior for all contingencies of interaction (in sickness and in health) over the lifetime of the object (till death us do part) [We1]. The folk wisdom that marriage contracts cannot be reduced to sales contracts is computationally expressed by interaction not being reducible to algorithms.

The quip that "everyone talks about object-oriented programming but no one knows what it is" is as true today as it was 20 years ago. "Knowing what it is" has proved elusive because of the implicit belief that "what it is" must be reducible to algorithms. Though object-based programming has become a dominant technology, its foundations are not well understood because attempts to express object behavior by TM models have proved unsuccessful. Interactive models provide a broader framework than algorithms for defining "what it is". Component-based software technology is even less mature than object-based technology: it is the technology underlying interoperability, coordination models, pattern theory, and the World Wide Web [We4]. Knowing what it is requires liberation from sequential object-based models.

The transition to object-oriented programming made procedural structured programming based on composing while statements and functional obsolete. Objects behavior cannot be compositionally expressed in terms of more primitive components. Structured programming for actions (verbs) can be formally defined by function composition, while structured object-based programming for composite objects (nouns) is modeled by design patterns that have no compositional formal specifications [GHJV]. As a consequence, the study of design patterns is a noncompositional art rather than a science.

Compositionality is a desirable property for formal tractability of programs that has led to advocacy of functional and logic programming as a basis for computation. But it limits expressiveness by requiring the whole to be expressible as the sum of its parts. Actual object-oriented programs and computer networks exhibit noncompositional emergent behavior. There are inherent trade-offs between formalizability and expressiveness that are clearly brought out by the expressive limitations of compositionality. Arguments in the 1960s that go-tos are considered harmful for formalizability can be paralleled by arguments in the 1990s that compositionality is considered harmful for expressiveness.

Go-tos and noncompositionality are harmful for formalizability but beneficial for expressiveness. An extension of this argumetn suggests that interactive computing agents are harmful for formalizability but beneficial for expressiveness. In this paper we have shown that finite computing agents are necessary in expressing object-oriented and distributed computing, and that they can be formalized by non-well-founded sets and coinductive models. The principle that things should be as simple as possible but no simpler suggests that TMs are too simple a model to completely capture computation and that IMs are necessary for a comprehensive theory of computational behavior.

Programming in the large (PIL) is not determined by size, since a program consisting of a sequence of a million addition instructions is not PIL. PIL is synonymous with interactive programming, differing qualitatively from programming in the small in the same way that interactive programs differ from algorithms. Embedded and reactive systems that provide services over time are PIL, while noninteractive problem solving is not PIL even when the algorithm is complex and the program is large.

The evolution of artificial intelligence from logic and search to agent-oriented programming is remarkably similar to the evolution of software engineering. This paradigm shift is evident in research on agents [Ag], on interactive planning and control [DW], and in textbooks that systematically reformulate AI in terms of intelligent agents [RN]. AI illustrates more clearly than software engineering that reasoning is an inadequate basis for modeling [We1]. Though logic is a form of computation, computation cannot be entirely reduced to logic. The goals of the Fifth-Generation Computing Project of the 1980s, which aimed to provide a logic-based framework for universal computation, were in principle unrealizable.

Open systems can be precisely defined as interactive systems: interactive models provide a tool for clas-

sifying forms of openness and for analyzing open-system behavior. Empirical computer science can likewise be precisely defined as the study of interactive systems [We3].

7. References

[Ab] Samson Abramsky, Semantics of Interaction, in *Semantics and Logic of Computation*, ed A. Pitts and P. Dibyer, Cambridge, 1997.

[Ac] Peter Aczel, Non Well-Founded Sets, *CSLI Lecture Notes #14*, Stanford, 1988.

[Ag] Phil Agre, Computational Research on Interaction and Agency, *Artificial Intelligence*, January 1995.

[BE] Allan Borodin and Ran El Yaniv, Online Computation and Competitive Analysis, Cambridge 1998.

[BM] Jon Barwise and Lawrence Moss, *Vicious Circles*, CSLI Lecture Notes #60, Cambridge University Press, 1996.

[CW] Luca Cardelli and Peter Wegner, On Understanding Types, Data Abstraction, and Polymorphism, *Computing Surveys*, December 1985.

[De] Michael Detlefsen, Philosophy of Mathematics in the Twentieth Century, in *Philosophy of Science, Logic, and Mathematics in the Twentieth Century*, Ed. S. Shanker, Routledge, 1996.

[Di] Edsger Dijkstra, *The Discipline of Programming*, Prentice-Hall, 1976.

[DW] Thomas Dean and Michael Wellman, *Planning and Control*, Morgan Kaufman 1991.

[EB] Encyclopaedia Britannica, Article on Realism

[Fe} Solomon Feferman, Kurt Godel, Conviction and Caution, in *Godel's Theorem in Focus*, Ed Shanker, Methuen, 1988.

[Fi] Paul Finsler, *Finsler Set Theory: Platonism and Circularity*, Eds. David Booth and Renatus Ziegler, Birkhauser, 1996.

[GHJV] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.

[Gi] Jean-Yves Girard, Linear Logic, *Theoretical Computer Science 50*, 1987..

[GW] Dina Goldin and Peter Wegner, *Persistent Turing Machines*, Brown Technical Report, 1998.

[GW2] Dina Goldin and Peter Wegner, Behavior and Expressiveness of Persistent Turing Machines, Brown Technical Report, May 1999.

[JR] Bart Jacobs and Jan Rutten, A Tutorial on Coalgebras and Coinduction, EATCS Bulletin 62, 1997.

[Kr] Saul Kripke, A Completeness Theorem in Modal Logic, *Journal of Symbolic Logic*, 1959.

[MH] Marco Forti and Furio Honsell, Set Theory with Free Construction Principles, Annali Scuola Normale Superiore di Pisa, Classe di Scienze, 1983.

[Mi] Robin Milner, Operational and Algebraic Semantics of Concurrent Processes, *Handbook of Theoretical Computer Science*, J. van Leeuwen, editor, Elsevier, 1990.

[Pa] Christos Papadimitriou, Games Against Nature, *Journal of Computer and System Sciences 31*, 1985.

[PR] Michael Prasse and Peter Rittgen, Why Church's Thesis Still Holds. Some Notes on Peter Wegner's Tracts on Interaction and Computability, *The Computer Journal*, Vol 41, No. 6, 1998.

[Pr1] Vaughan Pratt, Chu Spaces and Their Interpretation as Concurrent Objects, in *Computer Science Today: Recent Trends and Developments*, Ed. Jan van Leeuwen, LNCS #1000, 1995.

[PY] Christos Papadimitriou and Mihalis Yannakakis, Shortest Paths Without a Map, *Theoretical Computer Science* 84-1, July 1991.

[RN] Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*, Addison-Wesley, 1994.

[RS] Michael Rabin and Dana Scott, Finite Automata and their Decision Problems,

[Sc] Dana Scott, The Lattice of Flow Diagrams, Lecture Notes in Mathematics, Springer 1971.

[Tu1] Alan Turing, On Computable Numbers with an Application to the Entscheidungsproblem, *Proc. London Math Soc*., 2:42, pp. 230-265, 1936.

[Tu2] Alan Turing, Computing Machinery and Intelligence, *Mind*, 1950.

[We1] Peter Wegner, Why Interaction is More Powerful Than Algorithms, *CACM*, May 1997.

[We2] Peter Wegner, Interactive Foundations of Computing, *Theoretical Computer Science*, Feb. 1998.

[We3] Peter Wegner, Towards Empirical Computer Science, *The Monist*, Issue on the Philosophy of Com-

putation, January 1999, available at www.cs.brown.edu/people/pw.
[We4] Peter Wegner, Interactive Software Technology, *Handbook of Computer Science and Engineering*, Ed A. Tucker, CRC Press, 1996.
[We5] Peter Wegner, Tradeoffs Between Reasoning and Modeling, in *Research Directions in Concurrent Object-Oriented Programming*, Eds. Agha, Wegner, Yonezawa, MIT Press 1993.
[WG] Peter Wegner and Dina Goldin, Mathematical Models of Interactive Computing, Brown Technical Report, www.cs.brown.edu/people/pw
[WG2] Peter Wegner and Dina Goldin, Coinductive Models of Finite Computing Agents, Proc. Coalgebra Workshop (CMCS '99), Electronic Notes in Theoretical Computer Science, Volume 19, March 1999.