Coinductive Models of Finite Computing Agents

Peter Wegner

Brown University
Providence, RI. USA

Dina Goldin

U.Mass - Boston
Boston, MA, USA

Abstract

This paper explores the role of coinductive methods in modeling finite interactive computing agents. The computational extension of computing agents from algorithms to interaction parallels the mathematical extension of set theory and algebra from inductive to coinductive models. Maximal fixed points are shown to play a role in models of observation that parallels minimal fixed points in inductive mathematics. The impact of interactive (coinductive) models on Church's thesis and the connection between incompleteness and greater expressiveness are examined. A final section shows that actual software systems are interactive rather than algorithmic. Coinductive models could become as important as inductive models for software technology as computer applications become increasingly interactive.

1 Extensions of Expressiveness

The robust equivalence of Turing machines, the lambda calculus, and recursively enumerable sets has led to widespread acceptance of the thesis that Turing machines express the intuitive notion of computation. Because researchers assumed that questions of expressiveness of finite computing agents had been settled once and for all, they did not explore alternative models of computability, focusing instead on questions of complexity, performance, and design of algorithms. The hypothesis that interactive finite computing agents are more expressive than algorithms opens up a research area that had been considered closed, and requires reexamination of fundamental assumptions about models of computation.

We have shown [We1,WG] that interaction machines that provide services over time are more expressive than Turing machines that compute functions. The extension of expressiveness from algorithms to interaction is formalized by recent extensions of set theory and algebra [Ac,BM,JR]. Non-well-

This paper will be published in
Electronic Notes in Theoretical Computer Science, Volume 19
URL: www.elsevier.nl/locate/entcs

founded set theory models sequential interaction by formalizing the semantics of streams [BM]:

extension of computational expressiveness: algorithms → interaction
extension of set theoretic expressiveness: well-founded sets → non-well-founded sets
extension of algebraic expressiveness: algebras → coalgebras

Models of interaction, just like algorithmic models, can be described in complementary ways by sets and algebras. Non-well-founded sets are the interactive analog of recursively enumerable sets for Turing machines, while coalgebras play the role of the lambda calculus. Coinduction underlies non-well-founded set theory, coalgebra, and interaction as an abstract principle of definition, reasoning, and modeling. Coinduction is related to abduction, which infers inner system properties from their observed behavior [We3].

extension of mathematical expressiveness: inductive models → coinductive (abductive) models

The equivalent expressiveness of TMs, algorithms, computable functions, and formal systems is due largely to their common use of induction as a basis for system specification.

The Church-Turing thesis has the form "the formally definable notion X corresponds to an intuitive notion Y". It equates the intuitive notion of algorithmic computation with the formal notion of Turing computable functions from integers to integers:

Church-Turing Thesis: The intuitive notion of algorithms is formally expressed by Turing machines.
X = *algorithms,* Y = *Turing machines*

In section 5 we extend the Church-Turing thesis from algorithms to sequential and multi-agent interaction by extending both the intuitive notion X of computation and the associated formal notion Y:

E1: The intuitive notion of sequential interaction is formally expressed by non-well-founded sets.
X = *sequential (single-stream) interaction,* Y = *non-well-founded sets*
E2: General interactive computing is formally expressed by coinductive models (coalgebras).
X = *general (multi-stream) interaction,* Y = *coinductive models (coalgebras)*

The extensions $E1, E2$ of the Church-Turing thesis provide mathematical legitimacy for models of interaction. Non-well-founded set theory and coalgebras are coinductive extensions of inductive formal models of computation that express extension of the intuitive algorithmic model of computation.

Section 2 introduces interaction machines as stream-processing agents and observation equivalence (bisimulation) as a metric for interactive expressiveness. Section 3 introduces non-well-founded set theory and coalgebras and indicates how they model streams and labeled transition systems. Section 4 shows that eliminating initiality and replacing minimality by maximality ex-

plain the greater expressiveness of coinduction over induction. Section 5 examines metamathematical implications of computing, including the extension of Church's thesis, ontological commitment in mathematical models, and the role of incompleteness theorems. Section 6 examines software engineering, showing that interactive models provide a unifying framework for object-oriented and agent-oriented programming.

2 From Turing to Interaction Machines

Turing machines (TMs) are finite agents that noninteractively transform input into output strings by sequences of actions. 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 $F : S \times T \rightarrow S \times T$. TMs transform finite input strings $x \in T^*$ to outputs $y = M(x)$ by a finite sequence of computation steps that read a symbol i from the tape, perform a state transition $(s, i) \rightarrow (s', o)$, write a symbol o, and/or move the reading head one position right or left [Tu1].

Interaction machines (IMs) play a role in modeling finite interactive agents comparable to TMs for algorithms. *Sequential interaction machines* (SIMs) model sequential interaction, transducing input into output streams.

Sequential interaction machine: SIMs are state-transition machines $M = (S, A, m)$ where S is an enumerable set of states, A is an enumerable set of dynamically bound actions (stream elements), and the transition mapping m: $S \times A \rightarrow S$ maps state-action pairs into new states. Dynamic binding of inputs is explicitly indicated by rewriting m as a mapping $S \rightarrow P(A \times S)$ from states to action-state pairs that expresses *input nondeterminism* of actions A.

Expressiveness of finite agents is specified by *observation equivalence* and is measured by the ability of observers to distinguish agent behavior [WG]. TM behaviors consist of sets of input-output pairs, while SIM behaviors consist of sets of sequences of observations. Given two finite agents M_1, M_2, their behavior $B(M_1), B(M_2)$ is *distinguishable* relative to a class of observers (testers) if the exclusive-or $B(M_1) \oplus B(M_2)$ is nonempty. Members of this set are called *distinguishability certificates*. Finite agents are *equivalent* if they cannot be distinguished. Equivalence of two SIMs can be falsified by a finite distinguishability certificate (observation sequence) but cannot be finitely verified, illustrating Popper's point that empirical laws of physics can be empirically falsified but not verified [Po]. Interactive equivalence expressed by *bisimulation* [Mi] has greater distinguishing power than off-line equivalence.

Inputs of TMs and computable functions are completely specified at the start of a computation. Computable functions $f : X \rightarrow Y$ have a domain X of input strings (integers) and determine a unique $y = f(x)$ for each $x \in X$. Interactive behavior cannot be specified as transformation from a domain X to a range Y because streams are dynamically generated.

Interaction streams have the form $(a_1, o_1), (a_2, o_2), \ldots$, where output o_k is

computed from action a_k but precedes and can influence a_{k+1}. This *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, and control processes. Late (lazy) binding of interactive inputs (input nondeterminism) is more expressive than early (static) binding not because interaction has greater function transformation power but because the richer input domains cannot be inductively specified by strings.

Persistent Turing machines (PTMs) are a canonical model for sequential interaction that is as direct an extension of TMs as possible [GW].

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

Single interactions of a PTM correspond to TM computations, but its persistent state allows its semantics to be extended to define SIM behavior. A PTM determines a SIM (S, A, f), where S is the set of states (contents) of the persistent worktape, and A is the set of inputs (corresponding to initial TM inputs). The transtion function f is a mapping, $f : A \times S \rightarrow S \times O$, computed by the underlying TM. Elements of both S and A are finite at any given time but unbounded.

To understand the extra power of SIMs over TMs, it is helpful to consider the extra power of TMs over *finite lookup tables* (FLTs), which return a value when given the corresponding key. A single-step transition for a TM computation consists of performing table look-up of the new TM state and tape action based on the current state and tape character. Besides table look-up, the only additional work performed by a TM during each step consists of "remembering" the state and performing tape I/O.

FLTs may be considered as primitive finite computing agents whose expressiveness is that of propositional calculus. A TM computation consisting of a sequence of FLT lookups is more expressive than an FLT computation because its behavior is no longer finite. Similarly, Turing-computable single-step transitions are the primitive computation step of SIM computations. SIM computations are more expressive than those of TMs because sequences of dynamically supplied (externally controlled) TM computation steps allow finite agents to exhibit nonenumerable possible behaviors.

The informal argument that SIMs are not expressible by algorithms is surprisingly simple, since it depends only on the notion that stream elements are supplied dynamically.

Proposition: Computations of SIMs are not expressible by or reducible to computations of TMs.

Informal proof: TMs cannot be represented by finite inputs because any finite input can always be interactively extended. Transduction of streams cannot be modeled by transformation of strings. Streams do not have a last element, can be dynamically extended by unpredictable adversaries, and can use previous outputs in determining the next input. TM input strings have a predetermined finite length that cannot be changed once the computation has started.

A more refined argument shows that SIMs are more expressive than TMs for finite computations. Let k be the *length* (number of interaction steps) of an interactive computation, and let B_k be the class of behaviors that can be observed by observers who can make k or fewer interactions. We can show that, for all $k > 0, B_k \subset B_{k+1}$, while the behavior of TMs corresponds to B_1 [WG]. Computations are more expressive for $k+1$ interactions than for k interactions in the sense that they can make finer observational distinctions. Bisimulation [Mi] captures the greater observational distinguishing power of $k+1$ over k inputs.

SIMs that buffer the output of their first k interactions can be distinguished by $k+1$ but not by k interactions. Interactive questioners (Ken Starr) can elicit more information about a questioned subject (Clinton) by k+1 than by k follow-up questions. The game *20 Questions* further illustrates the power of follow-up questions in gaining information. TMs have the expressiveness of PTMs with $k = 1$: their behavior is limited to answering a single question (or a predetermined noninteractive sequence of questions). The expressiveness of sequential interaction has been examined in greater detail in [WG], both for question-answering and for the Turing test [Tu2].

Multistream interaction machines (MIMs) are finite agents that interact with multiple autonomous agents, like airline reservation systems or distributed databases that provide services to multiple autonomous clients. MIM behavior is not expressible by SIMs [WG], contrasting with the fact that multitape TMs are no more expressive than single-tape TMs. Input-output coupling of streams cannot be preserved under merging. We show [WG] that input-output coupling (serializability) cannot be preserved for delegation of subtasks and other forms of nonserializable behavior, proving that MIMs are more expressive than SIMs:

Input-output coupling of autonomous streams cannot be preserved under merging
MIMs express nonserializable behavior not expressible by SIMs

The restriction of MIMs to serializable behavior, just as the restriction of SIMs to noninteractive behavior, realizes tractability at the cost of problem-solving power. Adding autonomous streams (observation channels) to a finite agent increases expressiveness, whereas adding noninteractive tapes simply increases the structural complexity of predetermined inputs, and does not affect expressive 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. 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].

Collaboration, coordination, and management is modeled by MIMs but not

by SIMs or PTMs

The behavior of MIMs is harder to formalize than that of SIMs, and greater expressiveness of MIMs than SIMs is harder to prove than greater expressiveness of SIMs than TMs. MIMs support the behavior of nonserializable transactions and true concurrency [Pr], while SIMs support only serializable transactions and interleaving concurrency. The metric of distinguishability (observation nonequivalence) determines an expressiveness hierarchy for sequential interaction and greater expressiveness of MIMs than SIMs.

3 Non-Well-Founded Sets and Coalgebras

Though questions of expressiveness can be formulated and proved entirely in terms of machine-based notions, reformulation in terms of non-well-founded set theory and coalgebras lends mathematical legitimacy to interactive models.

A binary relation R on a set S is *non-well-founded* if there is an infinite sequence $b_I \in S$ for $i = 0, 1, 2, \ldots$, such that $b_i + 1 R b_i$ for all I, and is *well-founded* otherwise. Well-foundedness of sets is defined in terms of well-foundedness of the set membership relation $\in$; set is *well-founded* if the set membership relation over its structure is well-founded and is *non-well-founded* otherwise. Well-founded sets have only finite set-membership chains, while non-well-founded sets may have infinite set-membership chains.

Sets in traditional set theory are inductively constructible from atomic elements by *union*, *intersection*, *complement*, and *cross-product* constructors that construct sets with atomic elements. The *powerset* constructor constructs sets whose elements are also sets. *Zermelo-Frankel set theory* provides an axiomatic specification ZFC- of set theory that can be supplemented by the *foundation axiom* to obtain the inductively specifiable sets ZFC.

foundation axiom (FA): all sets are well-founded (inductively definable).

FA excludes non-well-founded sets, which are defined by 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 $m : S \rightarrow P(A \cup S)$ is a mapping that specifies for each $s \in S$ a set of constants in A and variables in S. A *non-well-founded set* can be defined as a set that is the solution of a system of flat set equations.

Equations of the form $m : S \rightarrow P(A \times S)$ whose right-hand sides are sets of ordered pairs can be expressed as flat equations by the transformation $(a, b) \rightarrow a, a, b$ [BM].

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

Non-well-founded set theory augments ZFC- with the *anti-foundation axiom* (AFA) to obtain ZFA. [BM] specifies AFA by the condition that all flat systems of equations have unique solutions.

anti-foundation axiom (AFA): Every flat system of equations has a

unique solution (ZFC- + AFA = ZFA).

ZFA is consistent if ZFC- is consistent and admits a larger class of sets (models a larger class of systems) than ZFC. The solution of the set equation $s = \{(0, s), (1, s)\}$ 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.

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 $m : S \rightarrow P(A \times S)$ maps states into subsets of action-state pairs. For example, a state with outgoing edges labeled a to s_1, b to s_2, and c to s_3 is represented by the mapping $s \rightarrow \{(a, s_1), (b, s_2), (c, s_3)\}$.

The mapping function m for systems of flat set equations is coalgebraic. *Coalgebras* are a model for interactive computing that parallels algebras as a modeling framework for algorithms. Algebraic computing steps specify incremental expression evaluation whose goal is to compute a value, while coalgebraic computing steps specify incremental system observation whose goal is to determine system behavior.

Algebras are structures $A = (S, m : F(S) \rightarrow S)$, 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 $m : F(S)\ rightarrow S$ is a value-preserving homomorphism 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 $CA = (S, m : S \rightarrow \Gamma(S))$, 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 $m : S \rightarrow \Gamma(S)$ as a behavior-preserving homomorphism from 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 are specified by non-well-founded sets.

Initial algebras are an inductively specified set of expressions, while final coalgebras are coinductively specified non-well-founded sets. The greater richness of non-well-founded over well-founded sets translates into greater computation power of coalgebras over algebras. 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.

Coalgebras that specify non-well-founded sets and model sequential interaction have mappings of the form $m : S \rightarrow P(A \cup S)$, corresponding to the AFA. It is shown in [BM] that mappings such as $m : S \rightarrow P(A \times S)$ of LTSs or mappings $m : S \rightarrow O \times P(A \times S)$ of automata are expressible as flat systems of equations. However, coalgebraic mappings can be more complex and model

some forms of multi-stream interaction.

Coalgebraic mappings of the form $S \rightarrow \Gamma(S)$ are "Markovian" in the sense that the new state depends only on the previous state. Non-Markovian mappings of the form $\Gamma'(S) \rightarrow \Gamma(S)$ can model some transaction systems (MIMs) whose next state depends on several previous states. We conjecture that interaction of overlapping operations with duration or multi-agent interaction may require still more complex mappings. Moreover, we believe that consistent extensions of set theory that admit larger sets than the non-well-founded sets are possible, corresponding to nonsequential and multi-stream finite agents [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 and may have a coalgebraic mapping function more complex than that of non-well-founded sets. The formalization of multi-stream interaction is a subject of future research beyond the scope of this paper.

4 From Induction to Coinduction

Induction determines a construction paradigm for definition, reasoning, and modeling characterized by an *initiality condition*, an *iteration condition* that allows new elements to be constructed (derived) from initial elements, and a *minimality condition* that only elements so constructed are definable:

inductive definition: initiality (base) condition, constructive iteration condition, minimality condition

construction paradigm: generate structures inductively from base elements

Induction is the basis for defining sets of strings, languages, formal systems, and computable functions. For example, the set A^* of all strings over an alphabet A is inductively defined by an iterative concatenation rule:

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

Grammars G define languages $L(G)$ over terminal symbols T with the aid of nonterminal symbols N as sets of strings inductively generated from an initial nonterminal $S \in N$ by generating rules (productions) P:

A language $L(G)$ over T is the set $L(G) = \{x \in T^* | S \Rightarrow x\}$, where $G = (N, T, S, P)$ is a grammar with nonterminals N, terminals T, initial symbol $S \in N$ and productions P. The derivation relation $\Rightarrow$ is the multi-step transitive closure of the single-step derivation relation defined by P.

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, where (1) axioms AX are theorems; (2) formulae derived from theorems by rules of inference RI are theorems; and (3) no other formulae are theorems.

Formal systems are inductive at two distinct levels. 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.

inductively defined domain (statics): the domain X is inductively defined
inductively defined computation (dynamics): the process of computation is inductively defined

Computable functions $f : X \rightarrow Y$ likewise make use of two levels of inductive definition. 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.

Coinduction determines a *deconstruction paradigm* that deconstructs composite structures into progressively more primitive ones. Non-well-founded sets are coinductively defined by a process of progressive decomposition into subsets that terminates finitely for well-founded (inductive) sets. Coinduction reverses the direction of iteration of an associated inductive process and replaces initiality with finality. Non-well-founded sets introduce a larger class of sets by eliminating finite termination, thereby eliminating initiality of dual inductive processes.

Coinduction models processes of *observation*. Elimination of finite termination corresponds to the fact that processes of observation can continue to reveal new knowledge about the observed objects indefinitely: hidden information is progressively approximated by processes that do not terminate. Coinduction reverses the direction of iteration, eliminates the initiality/finality condition, and replaces the minimality condition by a maximality condition:

coinductive definition: deconstructive iteration condition, maximality condition
observation paradigm: observe and transduce already existing constructed elements

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 $S = A \times S$. 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 $S = A \times S$ is the empty set, while the maximal fixed point is the set of all streams over S.

By expressing induction and coinduction in terms of more primitive concepts, we can separately consider the effect of reversing iteration, eliminating initiality, and replacing minimality by maximality in definition and reasoning processes. Finite coinductive termination, like initiality, is a closed-system requirement whose elimination models open systems. Elimination of initiality is related to shedding of the initial-state (and final-state) requirement in extending Turing machines to interaction machines. It removes the requirement of complete environment specification before the computation starts.

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:

the minimality principle is totalitarianism: everything is forbidden that is not allowed

the maximality principle is democracy: 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

The views of Church and Turing in the early 1930's were strongly influenced by the intense debates on mathematical foundations in the early 20th century, which were in turn influenced by Kant's earlier analysis of the distinction between necessary truths of mathematics and contingent truths of physics [De]. Church's interest in the "intuitive" notion of computing was due in part to the prominence of Brouwer's "intuitionism" and Hilbert's "formalism" as

paradigms of mathematical thought.

Coinductive models require reexamination of "foundations" for both mathematics and computing, suggesting new interpretations of both Brouwer's "intuitionist" belief that mathematical reasoning is based on inner intuitions, and of Hilbert's "formalist" belief that mathematical reasoning can be defined by formal logic. Coinductive thinking can be viewed as a new paradigm that replaces intuitionism and formalism by qualitatively new forms of reasoning. We prefer to view it as a strengthening of intuitionism and formalism that expresses stronger forms of intentionality and stronger (interactive) forms of the Turing test [We3]. Coinduction models stronger behavior of finite agents than traditional intuitionist or formalist models because of a stronger ontological commitment to the existence of mathematical objects.

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

5.1 From Formal Models to Intuitive Notions

Understanding of relations between formalisms and intuitive notions being formalized is a central goal both of Godel's work on completeness/incompleteness and of ChurchUs thesis. Theses that relate intuitive to formal models of computing can be 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. He recognized that the answer to such questions could not be definitive, but the equivalent expressiveness of alternative formalisms for computability appeared to provide strong evidence for the Church-Turing thesis.

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).

The Church-Turing thesis answers the question "What is the intuitive notion of computing that is expressed by TMs?", but not the question "What is the formal model that expresses the intuitive notion of computing?". In the early years of computing the intuitive notion of computing was identified with algorithms, and the two above questions were considered to have the same answer. As technology became increasingly interactive, and it was realized that algorithms could not express interaction, 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 the thesis to be extended.

We do not disagree with Church's thesis but claim that the robustness of Church-Turing models is due to their common inductive limitation 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, the formal notion of computability must be correspondingly broadened to coinductive specification 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.

The Church-Turing thesis was formalized in terms of computable functions $f : X \rightarrow Y$ from integers to integers. Domains X of integers model effective Turing computability but exclude interaction streams (section 2) whose domains are not inductively definable. Expressing domains as 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 streams. Stream mappings f : stream $\rightarrow$ 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

Godel's incompleteness result showed the impossibility of HilbertUs program of reducing mathematics to first-order logic [Fe] and by implication the impossibility of reducing the behavior of finite computing agents to logic [WG]. Completeness results establish the adequacy of syntactic formalisms for completely expressing semantics concepts. Church's thesis, that equates formal (syntactic) with intuitive (semantic) notions of computation, can be viewed as a completeness conjecture of TMs for an intuitive semantic notion of com-

putability.

Godel's incompleteness theorem can be viewed as a result about the weakness (incompleteness) of inductive reasoning for objective mathematical systems that include integer arithmetic. However, Turing's model of computation is "pre-Godelian" in its inductive (constructive) model of the notion of computation. The position that TMs completely express intuitive computability can be equated to Hilbert's position that first-order logic completely formalizes mathematics.

Godel took advantage of the fact that computable functions are inductively specifiable in his use of Godel numbering to enumerate the TMs. 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 and therefore inductively incomplete can be hard, but diagonalization is a useful tool in proving nonenumerability and 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 inductively enumerable
Diagonalization is a tool for showing that specific domains are not inductively enumerable

Interaction machines have nonenumerable possible behaviors and are therefore Godel incomplete. Well-founded sets are enumerable while non-well-founded sets are not. 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.

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 the behavior of any SIM and provides a model for non-well-founded set theory. However, the exploration of connections between linear logic and non-well-founded set theory 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 for every MIM there is a consistent coinductive axiomatic specification of set theory that expresses its behavior. 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.

In [We3] MIM behavior is related to quantum-theoretic nondeterminism and coinductive specification is extended beyond "hidden variables" to hidden active interfaces that cause nondeterministic behavior. However, further elaboration of the coinductive Church-Turing thesis is beyond the scope of this paper.

This thesis can be formulated as a completeness thesis for coinductive computation. Each of our three 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 based on circular reasoning 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 and Hilbert's formalism.

Though Hilbert in principle accepted the Platonic position that consistency (of a formalism) implies existence (of a model), he restricted formalism to inductive definition and reasoning, excluding consistent as well as inconsistent forms of 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.

Finsler's prescient work on set theory in the 1920's [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 1920's that non-well-founded set theory models interactive computing validates the view that conceptually consistent possible worlds 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 "consis-

tency implies existence" ($C \rightarrow E$), 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 ($C \rightarrow E$). Our analysis suggests that it may stem from the fact that $C \rightarrow E$ 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 $C \rightarrow E$ 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 $C \rightarrow E$ and Hilbert formalism.

Claim: $C \rightarrow E$, inductive formalism, and objective mathematics cannot be simultaneously true.

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

Hilbert: $C \rightarrow E$ + inductive formalism; inconsistency with objective mathematics was proved by Godel
Godel: objective mathematics + inductive formalism; explains Godel's rejection of $C \rightarrow E$
Cantor, Finsler: $C \rightarrow E$ + objective mathematics; implies rejection of inductive formalism

Hilbert's belief in $C \rightarrow E$ 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 $C \rightarrow E$. The realist, coinductive paradigm of mathematics corresponds to Cantor and Finsler's belief in $C \rightarrow E$ 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 $C \rightarrow E$ 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 computer architecture from mainframes to personal computers and networks, of software engineering from procedure-oriented to object-oriented and component-based systems, and of AI from logic-based to agent-oriented and distributed systems has followed parallel paths [We1]. According to the capsule history below, the 1950's through the 1970's were concerned with the development and refinement of algorithm technology for mainframes, sequential interaction became the dominant technology of the 1980's, while distributed interaction became the dominant technology in the 1990's:

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

Fig. 1. Parallel Extensions from Algorithms to Interaction

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

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 algorithms to interaction. 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 1 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. The domain-independent generality of SIMs and MIMs is both an advantage in providing a uniform modeling framework and a drawback in that general models provide little guidance for domain-specific applications.

The transition from the view that computing is primarily concerned with input-output transformations to the view that computing systems provide services over time arises in many different contexts. Services over time are specified by models of interaction that cannot be reduced to or expressed by algorithms or TMs. Algorithms are time-independent (instantaneous) episodes in the life-cycle of an interactive systems. 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.

Procedures and objects both determine a *contract* between providers and clients of a resource, but objects provide richer services to clients that can-

not be expressed by algorithmically specified procedures. Algorithms are like sales contracts, guaranteeing an output for every input, while objects are like 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.

Though object-based programming has become a dominant technology, its foundations are still shaky. Everyone talks about it but no one knows what it is. "Knowing what it is" has proved elusive because of the implicit belief that "what it is" must be defined in terms of algorithms. Interactive models have the liberating effect of providing 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. Knowing what it is in turn requires liberation from sequential object-based models.

Structured programming [Di] proved too weak as a model for program structure because the transition to object-oriented programming made procedural structured programming based on composing algorithms and while statements obsolete. Objects have behavior that cannot be compositionally expressed in terms of the behavior of their components. Structured programming for actions (verbs) can be formally defined by function composition, while structured programming for objects (nouns) is modeled by design patterns that have no compositional formal specifications [GHJV]. As a consequence, the study of design pattern methods of component composition is an 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 1960's that go-tos are considered harmful for formalizability can be paralleled by arguments in the 1990's that compositionality is considered harmful for expressiveness.

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 1980's, 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 classifying forms of openness and for analyzing open-system behavior. *Empirical computer science* can likewise be precisely defined as the study of interactive systems [We3].

References

[Ab] Samson Abramsky. Semantics of Interaction. *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.

[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.

[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.

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

[Po] Karl Popper. *The Logic of Scientific Discovery*, Harper, 1965.

[Pr] 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.

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

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

[Tu1] Alan Turing. On Computable Numbers with an Application to the Entscheidungsproblem. *Proc. London Math Society*, 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. *Communications of the ACM*, 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 Computation, Jan. 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`.