Mathematical Models of Interactive Computing

Peter Wegner and Dina Goldin

Draft, January 1 1999

Contents

Abstract: Finite computing agents that interact with an environment are shown to be more expressive than Turing machines according to a notion of expressiveness that measures problem-solving ability and is specified by observation equivalence. Sequential interactive models of objects, agents, and embedded systems are shown to be more expressive than algorithms. Multi-agent (distributed) models of coordination, collaboration, and true concurrency are shown to be more expressive than sequential models. The technology shift from algorithms to interaction is expressed by a mathematical paradigm shift that extends inductive definition and reasoning methods for finite agents to coinductive methods of set theory and algebra.

An introduction to models of interactive computing is followed by an account of mathematical models of sequential interaction in terms of coinductive methods of non-well-founded set theory, coalgebras, and bisimulation. Models of distributed information flow and multi-agent interaction are developed, and the Turing test is extended to interactive sequential and distributed models of computation. Specification of interactive systems is defined in terms of observable behavior, Godel incompleteness is shown for interaction machines, and explanatory power of physical theories is shown to correspond to expressiveness for models of computation. Our goal is to provide multiple perspective on interactive modeling from the viewpoint of interaction machines and mathematics, to persuade readers that interaction paradigms can play a significant role in narrowing the gap between theoretical models and software practice.

1. Introduction

Section 1 introduces concepts and models discussed in detail in the body of the paper. Section 1.1 reviews models of computation of finite agents, introducing the idea that finite stream processing agents that provide services over time are more expressive than Turing machines that transform strings. Section 1.2 defines models of sequential (single-stream) and distributed (multiple-stream) interaction machines, introduces an expressiveness hierarchy for sequential interaction, and shows that distributed finite agents are more expressive than sequential agents. Section 1.3 reviews mathematical extensions of set theory and algebra for modeling interaction. Section 1.4 shows that evolution from mainframes to personal computers and networks requires an extension of models of computation from algorithms to interaction.

1.1 Finite Computing Agents

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 determining the scope of theoretical computer science. Models of interaction challenge this dominance on the grounds that TMs are too weak to express interactive processes of finite computing agents. TMs model time-independent transformation of inputs into outputs but not interactive (reactive, embedded) services over time.

The Church-Turing thesis, which asserts that TMs and the lambda calculus express the intuitive notion of *effectively computable functions*, led to the widespread belief that behavior of actual computers could be expressed by TMs. Because researchers believed that questions of expressiveness of finite computing agents had been settled once and for all, they did not explore alternative notions of expressiveness. They focused instead on questions of complexity, performance, and design for the fixed notions of computability and expressiveness of Turing machines. The hypothesis that interactive finite computing agents are more expressive than algorithms opens up a research area that had been considered closed, requiring fundamental assumptions about models of computation to be reexamined.

Turing's seminal paper [Tu1] was not intended to establish TMs as a universal model for computing but on the contrary to show undecidability and other limitations of TMs. Turing actually distinguishes between automatic machines (a-machines) and choice machines (c-machines) in [Tu1] and excludes c-machines in fashioning his model of computation, presumably because they do not conform to his deliberately limited notion of computability. TMs model "automatic" computation, while *interaction machines* (IMs) extend the notion of what is computable to nonautomatic (interactive) computation. IMs can model TMs with oracles, which Turing showed to be more powerful than TMs in his Ph.D. thesis [Tu2].

The irreducibility of interactive systems to algorithms was noticed by [MP] for *reactive systems*, by [Mi] in the context of *process models*, and by many other researchers. We identify interaction as a domain-independent and language-independent cause of greater computational expressiveness, viewing interaction as normal behavior that needs to be analyzed rather than as exceptional unanalyzable behavior. Interactive models cannot be specified or analyzed by tools of inductive mathematics or first-order logic. Fortunately coinductive models of non-well-founded set theory and coalgebra [Ac, BM, Ru1] provide models for interaction. The paradigm shift from algorithms to interaction can be handled by a mathematical paradigm shift from inductive to coinductive models described in sections 1.3 and section 3.

Interaction machines for finite agents have a role comparable to that of *Turing machines (TMs)* for algorithms. Sequential interaction architecture is modeled by *sequential interaction machines (SIMs),* while *persistent Turing machines (PTMs)* are a canonical model for sequential interaction whose role parallels that of TMs for algorithms. Distributed interaction architecture is modeled by *multi-agent interaction machines (MIMs). Expressiveness* of finite agents is specified by *observation equivalence*, and is measured by the ability of observers to distinguish agent behavior. SIMs are shown to be more expressive than TMs and MIMs are in turn shown to be more expressive than SIMs.

Two finite agents have distinct behavior relative to a class of observers (testers) if there is an observation, called a *distinguishability certificate*, that distinguishes them. Two finite agents are equivalent if they cannot be distinguished. Equivalence can be falsified by a finite distinguishability certificate but cannot be finitely verified, as shown in the context of philosophy of science by Karl Popper [Po].

We show in section 2 that PTMs are more expressive than TMs for finite computations. Let the length of an interactive computation be its number of interaction steps and let PTM(k) be the class of PTMs capable of computations of interactive length k. We show that PTM(k+1) has greater distinguishing power than PTM(k) for all k>0, while the distinguishing power of TMs is PTM(1). Once the idea of greater expressiveness is accepted, we can exhibit an expressiveness hierarchy for sequential interaction and show that multi-agent machines (MIMs) are more expressive than single-agent machines (SIMs).

Greater distinguishing power implies both the ability of observers to make finer distinctions and the ability of finite agents to exhibit a greater range of behavior (solve a larger class of problems). Greater expressiveness defined by distinguishing power of observers corresponds to greater problem solving ability by handling a larger domain of inputs (observations) from the environment.

Late (lazy) binding of interactive inputs is more expressive than early (static) binding not because interaction has greater function transformation power but because of richer input domains that cannot be inductively specified by strings. Greater distinguishability by observers implies a greater range of behavior by agents, which in turn implies a greater input domain for interactive streams over predetermined strings that is formalized by non-well-founded sets (section 1.3). Computation power is extended beyond Turing computability by extending input domains from strings to streams.

Domain elements of functions computable by TMs are completely specified at the start of a computation: TMs compute outputs noninteractively from their inputs. Computable functions f: X -> Y, have a domain X of input strings (integers) and determine a unique y = f(x) for each x∈X. Interactive behavior cannot be specified as transformation from a domain X to a range Y because *stream* elements are dynamically generated (section 3.1). Streams have the form (a_1,o_1), (a_2,o_2), ... , 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.

Stream transductions are not functions from integers to integers because streams have incrementally generated domains not representable by integers. The computational extension from strings to streams is mathematically expressed by an extension from inductive to coinductive definition and reasoning principles. Streams and strings have inductively specified domains while streams have coinductively specified domains that are not enumerable. Turing machines are mathematically modeled by induction-based principles of constructive mathematics while interaction machines are modeled by an emerging coinductive paradigm [BM] that expresses observation (testing) of objects in an already constructed world (section 5.4).

Question-answering has been used both by logicians [Kl] and by Turing [Tu3] as a framework for investigating the expressiveness of models. Finite agents that transform strings can answer only enumerable classes of inductively specifiable questions, while finite stream processing agents can distinguish among nonenumerable questions (situations) [BM]. For example, finite agents can distinguish among the nonenumerable real numbers in the sense that any pair of distinct reals, represented by infinite binary strings, has a finite distinguishability certificate. Nonenumerability of streams is shown mathematically in section 3 by modeling streams as non-well-founded sets.

The idea that interaction is not expressible by or reducible to algorithms was proposed by the first author in 1992 at the closing conference of the fifth-generation computing project in the context of logic programming [We4]. Reactiveness of logic programs, realized by committed-choice at each interaction step, was shown to be incompatible with logical completeness, realized by backtracking. The 5th-generation project's failure to achieve its maximal objective of reducing computation to first-order logic was attributed to theoretical impossibility rather than to lack of cleverness of researchers. The implication that success could not have been achieved by a twenty year extension or substantial further research demonstrated the practical impact of impossibility results in justifying strategic decisions to terminate projects.

The irreducibility of interaction to algorithms and of computation to first-order logic reinforces the view that tools of algorithm analysis and formal methods cannot, by themselves, solve the software crisis. Software tools for design patterns, life-cycle models, embedded systems, and collaborative planning are

not modeled by algorithms and their behavior cannot inherently be expressed by first-order logic. Fred Brooks' claim that there is no silver bullet for systems can be proved if we define "silver bullet" to mean "algorithmic or first-order logic specification". Computable functions are too strong an abstraction to model the complete behavior of actual computing systems, sacrificing the ability to model interaction and time to realize tractability. New classes of models are needed to express the technology of interaction, since software technology has outstripped algorithm-based models of computation (section 1.4).

The impossibility of modeling interaction by first-order logic or traditional set theory suggests that mathematical models of computation are inherently limited and that a choice between formal algorithmic models and unformalizable interactive models is inevitable. Fortunately a mathematical framework for interactive modeling has emerged [Ac, BM] that makes such a choice unnecessary. Negative impossibility results useful in avoiding expenditures on unsolvable problems lead to positive formal models of computation that provide a framework for a technology of interactive computing (section 3).

1.2 Interaction Machines

TMs are finite agents that noninteractively transform inputs into outputs by sequences of actions.

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 -> SxT. TMs transform finite input strings $x \in T^*$ to outputs y = M(x) by finite computations consisting of steps that read a symbol i from the tape, perform a state transition (s,i) -> (s',o), write a symbol o, and move the reading head one position to the right or left [Tu1].

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. In contrast, IM responses have history-dependent effects that depend on a persistent state and on interactively generated streams that cannot be represented by strings. The shift from algorithms to sequential interaction is a shift from string transformation to stream transduction. SIM specifications are syntactically like TM specifications without an initial state, but their semantics of computation (section 2) differs from that of TMs.

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 specified by streams, and the transition mapping m: SxA -> S maps state-action pairs into new states.

The informal argument that SIMs are not expressible by algorithms is surprisingly simple, depending only on the notion that stream elements are dynamically supplied. We present a more refined argument in section 2 showing that SIMs are more expressive than TMs for finite computations, and formalize these arguments in section 3 in the context of non-well-founded set theory.

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

Informal proof: TMs that noninteractively transform input to output strings cannot express computations on dynamically generated streams. 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.

SIMs have nondeterministic inputs: *input nondeterminism* differs fundamentally from *output nondeterminism* of nondeterministic automata (multiple transitions for a given state-input pair). SIMs define a language-independent model that expresses sequential interaction in question-answering situations, control applications, sequential object-oriented languages and single-user databases. Persistent TMs provide a canonical model for SIMs by a minimal extension of TMs to multitape machines with a persistent working tape and an enumerable number of states (finite at any given time but not bounded). Bisimulation captures the expressiveness of SIMs and PTMs by an equivalence relation (section 3.4).

The length k of a PTM computation is the number of its interaction steps: each interaction step corre-

sponds to a TM computation, but its effect is history-dependent and its input can depend on earlier outputs and other dynamic events. We show in section 2.1 that, for all k>0, PTM computations are more expressive for k+1 interactions than for k interactions in the sense that they can make finer observational distinctions. PTMs 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 also 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).

Observed expressiveness depends both on the behavior of observed systems and on the observation power of observers. Behavior cannot be observed unless observers are able to perceive it. When observers (testers) are limited to single interactions then observable behavior is limited to TM behavior since interactive behavior is unobservable. Greater expressiveness was not noticed largely because testing of input-output behavior is incapable of observing interactive distinguishability certificates.

Multi-stream distributed interaction machines (MIMs) are finite agents that interact with multiple autonomous agents, like airline reservation systems that interact with multiple travel agents or distributed databases that provide services to multiple autonomous clients. MIMs are shown to be more expressive than and not reducible to sequential interaction, contrasting with the fact that multitape TMs are no more expressive than single-tape TMs. MIMs allow higher-level behavior such as collaboration and coordination to be precisely modeled. The greater expressiveness of MIMs over SIMs captures the greater problem-solving power of senior managers and CEOs over workers that perform sequences of interactively assigned tasks. Adding autonomous streams (observation channels) to a finite agent increases expressiveness, while adding noninteractive tapes simply increases the structural complexity of predetermined inputs, and does not increase expressive power.

Though there is strong evidence that MIM behavior is not expressible by SIMs, MIM behavior is harder to formalize and greater expressiveness is harder to prove. MIMs support the behavior of nonserializable transactions and true concurrency [Pr], while SIMs support only serializable transactions and interleaving concurrency. The proof that MIMs are more expressive than SIMs (section 3.2) shows that input-output coupling of streams cannot be preserved under merging. We show that input-output coupling (serializability) cannot be preserved for delegation of subtasks and other forms of nonserializable behavior.

input-output coupling of autonomous streams cannot be preserved under merging

Concurrent execution of MIMs is shown to be dual to sequential execution of SIMs. Extension of algorithmic to interactive concurrency for MIMs parallels the extension of algorithmic to interactive sequential computation of SIMs. The extension from SIM interaction between two agents to MIM interaction among three or more agents parallels the extension in physics from the tractable two-body problem to the intractable 3-body and n-body problems.

Distributed systems express regularities of behavior that carry information about remotely occurring events to interfaces that are immediately observable. A mathematical model of distributed information flow developed in [BS] and examined in section 3.1 describes noninteractive distributed behavior. MIMs support distributed information flow among interfaces in the context of interaction.

Observed behavior of finite agents can be meaningfully specified only relative to an observer, both for physical and computational systems. TM observers can observe only behavior for single I/O pairs, while SIM observers can observe the behavior of interactively supplied sequences of inputs that can depend on dynamically occurring events in the environment. For MIMs, finite agents can observe only sequential behavior of single streams. Complete behavior at all interfaces of a distributed system, such as an airline reservation system or nuclear reactor, cannot be observed by finite agents, though it could in principle be observed by omniscient distributed observers (God). Since there are forms of behavior that can be specified but not physically observed, there is a distinction between specifiable behavior and observable behavior: in particular, the complete behavior of MIMs is specifiable but not observable, leading to *subjective nondeterminism* due to limitations of observers related to quantum nondeterminism (section 4.3).

Parallels between computational and physical models explored in [We3] establish substantive connections between empirical computer science and models of observation of relativity and quantum theory. In particular, interactive models provide a stronger formulation of Einstein's argument that nondeterminism is due to incomplete observability, based on hidden interfaces rather than hidden variables. Einstein's intuition that God does not play dice and that observed nondeterminism is due to weaknesses of our model of observation rather than inherent in nature is modeled by primary observers who have no control over "random noise" due to secondary observers.

These parallel extensions from algorithms to interaction are captured by a mathematical paradigm shift that provides a foundation for sequential models of interaction, and requires new modes of mathematical thinking that will play an increasingly important role in undergraduate and graduate education. A key element in this paradigm shift is the extension from inductive definition and reasoning about enumerable sets to coinductive definition and reasoning about nonenumerable sets.

1.3 Mathematics of Interactive Computing

The extension in computational expressiveness from algorithms to interaction, which cannot be formalized in first-order logic and was therefore considered unformalizable, is modeled and thereby formalized by a mathematical extension in expressiveness of set theory and algebra. Models of interaction can be described in complementary ways by non-well-founded set theory and coalgebra, just as algorithmic models have complementary descriptions by recursviely enumerable sets and the lambda calculus.

extension of set theoretic expressiveness: well-founded sets -> non-well-founded sets

extension of algebraic expressiveness: algebras -> coalgebras

extension of computational expressiveness: algorithms -> interaction

The set-theoretic extension can be defined in terms of axiomatic set theory by circular axioms [BM] that admit sets which are not inductively constructible. Zermelo-Frankel set theory with the *foundation axiom (FA)* limits admissible sets to those inductively constructible from primitive sets by set operations of union, intersection, and pwer set construction. It can be consistently extended by replacing the FA by the coinductive *anti-foundation axiom (AFA)*, which admits all sets that are unique solutions of set equations of a certain form [BM]. This method of extending mathematical domains of sets is similar to algebraic domain-extension methods that introduce rationals as solutions of linear equations with integer coefficients. The AFA extends domain of sets to include non-well-founded sets that model streams and sequential interaction. Non-well-founded set theory provides a consistent formal model for sequential interaction that formalizes stream behavior, and a denotational semantics for sequential interactive models.

foundation axiom: inductively definable sets of traditional (constructive) set theory

antifoundation axiom: coinductively definable sets of non-well-founded (interactive) set theory

The inadequacy of TMs and first-order logic as models of actual computers is due to the weakness of induction as a definition and reasoning principle for finite agents. The equivalent expressiveness of TMs, algorithms, computable functions, and formal systems is due largely to their common reliance on induction (section 3.1). Coinductive reasoning is related to *abduction*, which infers inner properties of systems from their observed behavior [We3].

extension of mathematical expressiveness: inductive models -> coinductive (abductive) models

Induction is a technique for definition and reasoning over enumerable linear or partially ordered structures such as integers, strings, trees, and theorems derivable from axioms. Coinduction is a technique for definition and reasoning over nonenumerable circular structures such as graphs, streams, and empirically observable phenomena. TMs are inductively defined string processing machines, while interaction machines are coinductively defined stream processing machines. Inductively defined well-founded sets have enumerable elements, while coinductively defined non-well-founded sets, such as streams, may have nonenumerable elements.

Induction is specified by an *initiality condition*, an *iteration condition* that allows new elements to be

derived from initial elements, and a *minimality condition* that only elements so derived can be considered. Initiality and minimality limit the expressive power of induction: elimination of initiality together with replacement of minimality by maximality yields coinductive models.

inductive definition: initiality (base) condition, iteration condition, minimality condition
construction paradigm: create and transform structure generated inductively from base elements
coinductive definition: iteration condition (circularity condition), maximality condition
observation paradigm: observe and transduce already existing constructed elements

By expressing induction and coinduction in terms of primitive constituents, we can separately consider the effect of eliminating inititiality and replacing minimality by maximality on definition and reasoning processes. Initiality is a closed-system requirement whose elimination makes it possible to model open systems. Elimination of initiality in mathematical models is related to shedding of the initial-state requirement in extending TMs to SIMs and PTMs and to focusing on rules of inference independently of axioms in formal systems. Elimination of initiality allows modeling of systems without preconceptions about the nature of their interaction with observers in the environment. In particular, predetermined static environments required by initiality can be generalized to incrementally supplied dynamic open environments.

The minimality condition "everything is forbidden that is not allowed" specifies a smaller class of things than the more permissive maximality condition "everything is allowed that is not forbidden". Minimality of behavior is associated with maximality of constraints on behavior, while maximality of behavior is associated with minimality of constraints. This contravariance of constraints and behavior arises in type theory and inheritance, and is the basis of duality results in physics and computing [Pr]. Minimality is modeled by least fixed points while maximality is modeled by greatest fixed points.

Set theory and algebra are two complementary modeling techniques that deal with the end-product and the process of modeling. Set theory specifies denotations, while algebra specifies the operational semantics of processes that transform denotations. The set-theoretic extension of denotations from well-founded to non-well-founded sets is paralleled algebraically by extension from algebraic to coalgebraic models of computation [Ru]. Algebras map expressions into representative elements of a value class, while coalgebras map systems into representative elements of a behavior class. Algebraic equivalence classes of expressions mapped into a given value are enumerable, while coalgebraic equivalence classes of systems mapped into a given behavior are nonenumerable (sets of programs with a given behavior cannot be enumerated).

[BM] suggests that the reluctance of logicians to consider coinductive models based on circular reasoning stems primarily from the failure to distinguish between inconsistent forms of circular reasoning that led to the paradoxes of set theory and consistent forms of circular reasoning. However, an additional reason was the strong influence on logic of Brouwer's intuitionism and Hilbert's formalism, which caused logicians to focus on properties of formalisms (proof theory) and to deny existence to independent concepts that formalisms aim to model. Constructive mathematics accords existence only to things that can be constructed, and even nonconstructive formalists restrict existence to truths specifiable by induction from inductively specifiable axioms. Both exclude concepts with nonenumerable elements or situations like the real numbers or the real world. Such concepts are unformalizable by enumerable inductive models, but can be formalized by coinductive (circular) reasoning.

Finsler remarkable work on set theory in the 1920s [Fi] showed the consistency of non-well-founded set theory and anticipated Godel's incompleteness result, but was largely ignored because it did not conform to the formalist ontology of Russell, Hilbert, and Tarski. Finsler adopted the viewpoint that concepts exist independently of formalisms in which they are expressed, influenced by Cantor's model of the real numbers. We refer to this viewpoint as an *empiricist ontology* of mathematics, since it accords existence to all consistent possible worlds, including the real world. Intuitionism, formalism, and empiricism can be classified by their degree of commitment to the existence of mathematical objects, embodying progressively stronger forms of ontological commitment that admit richer classes of semantic domains:

intuitionism (Brouwer): existence requires (inductive) constructibility (minimal ontology)
formalism (Hilbert): existence (of an interpretation of a formal system) implies consistency

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

The extension from intuitionist to formalist ontology rests on according a first class status to existential quantification, independently of constructibility. This approach was adopted in [CW] to model abstract data types with hidden variables. More research is needed on distinguishing intuitionist and formalist ontologies, but we focus primarily on the further extension from formalist to empiricist ontology.

The paradigm shift from inductive to coinductive definition and reasoning determines an ontological paradigm shift from constructive and formalist to empiricist principles of modeling, that can model observation of computational and physical systems. Constructive inductively-defined ontologies are specified by least fixed points, while empiricist coinductively-defined ontologies are specified by greatest fixed points. The stronger maximal ontology admits interactive and time-dependent behavior for finite agents while minimal ontology admits only noninteractive behavior. The strong connection between circular reasoning and empiricist models of actual and possible worlds is explored in [BM].

Though Platonism questions the existence of the real world, asserting that concepts are more real than physical objects, there is a paradoxically close connection between empiricist and Platonist mathematical models. Empiricist models accord reality to all consistent concepts and therefore accord existence to all consistent possible worlds, including the real world. Finsler argued that since non-well-founded set theory was consistent, non-well-founded sets existed independently of whether they had useful models. The discovery sixty years later that non-well-founded set theory models interactive computing provides an incentive for reevaluating Finsler's work, and validates the empiricist (Platonic) approach of according existence to conceptually consistent possible worlds independently of their formalizability or constructibility. Tensor analysis is another example of an abstract mathematical technique that found uses in relativity theory, linear logic, and a variety of other areas of applied mathematics many years after it was initially developed.

Godel's incompleteness result showed the impossibility of Hilbert's program of reducing mathematics to logic and by implication the impossibility of reducing computing to logic. In spite of this, Turing's model of computation is in the pre-Godelian constructive tradition. The computable functions are inductively specifiable (section 3.1) by first-order logic and therefore not able to completely specify arithmetic over the integers. Godel's incompleteness result follows from the folk theorem that first-order logic can model only enumerable semantic domains because its number of theorems are enumerable. The diagonalization argument that forms the basis of the incompleteness proof shows that arithmetic over the integers has a nonenumerable number of true assertions that cannot be modeled by any enumerable set of theorems.

Godel incompleteness can be viewed as a proof that constructive mathematics is too weak to express formalist models of arithmetic over the integers. This is sufficient to prove that interactive models are incomplete (section 5.2), since interactive models can certainly express all formalist models. Further refinement of incompleteness results to show that interactive models are strictly richer than formalist models and to distinguish among levels of expressiveness of interactive models is alluded to in section 5.2 but is beyond the scope of this paper.

Godel's constructive demonstration of incompleteness of first-order logic by arithmetization of metamathematics (Godel numbering) was hailed as a seminal result by formalist logicians, while Finsler's prior result that formalisms are incomplete in expressing concepts, because we can conceive nonenumerable sets while formalisms express only enumerable sets, was not considered significant. Godel's arithmetization of metamathematics allows mathematical theories (and TMs) to be expressed by arithmetic in an incredibly simple and precise way. But Godel then shows that such a formalization is inadequate because it is incomplete. TMs provide the same simple and precise model for computing that Godel provided for mathematics, but are limited in their modeling power for the same reason that incompleteness limits the power of Godelian (arithmetized) mathematical models. Arguments that interactive models are more expressive and more appropriate for modeling the real world parallel those of Finsler in the 1920s.

Minimality is a property of constructive paradigms for definition and reasoning that characterize both algorithmic computation and constructive mathematics, while maximality is a property of empirical observation paradigms for describing observed behavior in an already constructed (already existing) world.

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**

Interactive models show the limitations of formalist inductive mathematics and have contributed to a mathematical paradigm shift that reestablishes the importance of independently given conceptual domains in mathematical modeling. Though philosophical questions concerning the foundations of mathematics are outside the scope of this paper, they are relevant to understanding the relation between formally specified finite interactive agents and their nonformalized environment. We briefly return to these questions in discussing the antifoundation axiom in section 3.2, and the interactive Turing test in section 4.4.

1.4 Interactive 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 have followed parallel paths [We1]. According to the capsule history below, 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.

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: 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 paradigm shift from algorithms to interaction (section 2). The extension from sequential to distributed interaction architectures requires a further fundamental paradigm shift in models of computation (section 4). SIMs formalize the shift from algorithms to sequential interaction architecture while MIMs formalize 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 genrality of SIMs and MIMs is both an advantage in providing a uniform modeling framework and a draw-

back in that general models provide little guidance for domain-specific applications.

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 1: Parallel Extensions from Algorithms to Interaction

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 are not reducible to or expressible by algorithms or Turing machines. Algorithms are time-independent (instantaneous) episodes in the life-cycle of an interactive systems. The one dimensional quanitative performance metric of algorithmic complexity becomes the multidimensional qualitative performance metric of quality of service (QoS), which is becoming 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 cannot 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 for defining "what it is" than algorithms. 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 technology was the basis of an attempt to formalize software engineering associated with Dijkstra and other researchers [Di]. However, it ultimately proved too weak as a model for program structure because the transition to object-oriented programming made procedural structured programming based on composing programs with while statements and procedures 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.

Though compositionality is a desirable property for formal tractability of programs, and has led to advocacy of functional and logic programming as a basis for computation, 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 argu-

ments in the 1990s by arguments 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 instruction 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 that in 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, 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]. The interdisciplinary connections of interactive models to empirical methods of physics are examined in section 4 and 5.

2. Models of Sequential Interaction

Section 2.1 develops SIMs as a model for sequential interaction and persistent Turing machines (PTMs) as a canonical model for SIMs. Section 2.2 defines expressiveness in terms of observation equivalence and shows that there is a hierarchy of levels of interactive expressiveness beyond that of Turing machines. Section 2.3 examines relations between observers and observed systems.

2.1 Single-Stream (Sequential) Interaction Machines (SIMs)

Sequential interaction is expressed by SIMs, which 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 specified by streams, and the transition mapping m: SxA -> S maps state-action pairs into new states.

The formal difference between sequences of noninteractive computation steps and sequences of interactive actions is expressed by strings versus streams. The late nondeterministic binding of stream actions can be explicitly specified by an equivalent curried form of the SIM mapping:

m: S -> (A -> S) -- map states to functions from actions to states

This curried form of m explicitly expresses input nondeterminism by associating each state with a function from dynamically supplied actions to states. The following power set form of the mapping m makes input nondeterminism even more explicit:

m: S -> P(AxS) -- maps states into sets of state-action pairs (a,s')

Each state s∈ S is associated with a subset of action-state pairs that specifies possible next actions. Selection among possible next actions is determined nondeterminisitcally when the input arrives. Each step (transition) of a SIM corresponds to a TM computation, but SIM behavior for a sequence of steps cannot be described by a TM.

The mapping m: S -> P(A, S) is used to specify non-well-founded sets that model labeled transition systems (section 3.2) and for coalgebraic mapping, establishing that streams are specifiable as non-well-founded sets and that SIM computations are coalgebraic. SIMs with output nondeterminism (that extend nondeterministic TMs to interaction) can be easily defined but are not considered here.

Sequential interaction supports on-line processing of sequences of actions not known in advance, where each action must be completed before the next one is processed. On-line sequential interaction is the basis for a variety of applications such as reinforcement learning [SB], computational geometry [CT], and question answering (section 2.3). Our goal is to provide a domain-independent model for sequential interactive applications that captures the behavior of domain-independent object-oriented and component-based formalisms and can be specialized to specific domains.

Examples of sequential interaction: data abstractions, sequential objects, embedded reactive systems, drivers who adapt to road conditions, lecturers who adapt to their audience, questioners who can ask follow-up questions, two-person games and negotiation.

Persistent Turing machines (PTMs) specialize SIMs to a particular inner architecture that minimally extends Turing machines. PTMs extend TMs with a persistent worktape, eliminate the intitial state requirement of TMs, and have a history-dependent state represented by the content of the worktape. The behavior of any SIM can be expressed by a PTM [GW], just as the behavior of any machine for computing algorithms can be expressed by a TM. PTMs are a canonical model of SIMs designed to play a role for sequential interaction corresponding to that played by TMs for algorithms.

Persistent Turing Machines: A PTM = (S, A, f) is a multitape TM with a persistent work tape whose content is preserved between interactions. S, the set of PTM's *states*, consists of the states of the persistent worktape (i.e., its contents); A is the set of PTM's *actions*, or inputs; f is a *mapping*, f: AxS -> SxO, computed by the underlying TM. Both S and A are finite at any given time but unbounded.

Example: Answering Machine. An answering machine $\mathcal{A}$ is a triple ($S_{\mathcal{A}}$, $A_{\mathcal{A}}$, $f_{\mathcal{A}}$), whose states $S_{\mathcal{A}}$ represent the contents of the message tape. $\mathcal{A}$'s actions $A_{\mathcal{A}}$ are of three types: *record* X, *playback*, and *erase*, where X is any finite string. $\mathcal{A}$'s computation mapping $f_{\mathcal{A}}$ is defined as follows:

$f_{\mathcal{A}}$(*record* X, Y) = (*ok*, YX); $f_{\mathcal{A}}$(*playback*,X) = (X,X); $f_{\mathcal{A}}$(*erase*,X) = (*ok*,ε)

PTMs process sequences of actions, each of which is a single TM input. PTM *observations* are the resulting sequences of input-output pairs. Given a PTM *M* with a computation mapping f_M, an observation of *M* of length k is a sequence of k input-output pairs: $O_M = (a_1, o_1), (a_2, o_2), \ldots, (a_k, o_k)$, where a_i is in A, such that $f_M(a_1,w_o) = (o_1,w_1)$; $f_M(a_2,w_1) = (o_2,w_2)$, etc. Here, w_i is the contents of the persistent work tape, starting with the *initial* contents w_o. Preservation of the worktape (state) between interactions expresses history dependence by persistence. w_o is not necessarily empty, so the initiality condition is relative to a given observation.

Example: Answering Machine observations. The following is an observation of $\mathcal{A}$ of length 3:

(*playback*, `hello`), (*record* `hi`, *ok*), (*playback*, `hello hi`)

The *behavior* of a PTM, B(PTM), consists of all its finite observations. So, the observation above is in B($\mathcal{A}$), whereas the following one is not:

(*playback*, `hello`), (*record* `hi`, *ok*), (*playback*, `hello hello`)

The computation mapping of a PTM can be contrasted with the transition mapping of a TM, which is also of type SxA -> SxO. For TMs, S and A are finite sets of states and alphabet symbols respectively, and O is the TM's output at a single computation step, either writing a symbol at the current head position or moving the head. As a result, a TM's transition mapping is finite, and can be represented explicitly.

The abstraction from TM's character-based individual transitions to its string-based computation is paralleled by the same abstraction for PTMs, but at a higher level of granularity. PTM mappings determine second-order as opposed to first-order transitions. A PTM's input is a stream whose string inputs correspond to TM inputs, and whose computational steps correspond to TM computations. Whereas TMs specify input-output behavior for single algorithms, SIMs specify behavior of a sequence of algorithms.

2.2 Observation Equivalence and Expressiveness

We introduce observable behavior, observation equivalence, and an observation-based metric for interactive expressiveness, and show that according to this metric PTMs are more expressive than TMs. The observable behavior B(TM) of TMs is specified by single I/O pairs, while observable behavior B(SIM) of

SIMs is specified by streams of I/O pairs.

B(TM) = {(i,o) | where o = f(i) and f is a computable function}

B(SIM) = {(i_1,o_1), (i_2,o_2), ... | o_k is computed from i_k and precedes the generation of i_{k+1}}

Streams (a_1,o_1), (a_2,o_2), ... of a SIM are called *input-output (I/O) streams*. The phenomenon that o_k follows a_k and precedes $a_{(k+1)}$ is called *input-output (I/O) coupling*. Each input-output pair of an I/O stream is an atomic transaction that must be completed before processing the next input. I/O coupling, which determines *temporal granularity* and *temporal modularity*, is the key to greater interactive expressiveness. For sequential interaction, (a_i,o_i) pairs are serializable transactions modeled as instantaneous actions. Multi-agent interaction supports nonserializable transactions, actions with duration, and true concurrency [Pr]. Greater expressiveness of multi-agent over sequential interaction is shown by the impossibility of preserving sequential I/O coupling when merging autonomous streams (section 4.2.).

Observers of systems are specified by the class of observations they can make. The expressiveness of observers for a class C of systems is modeled by their ability to make observational distinctions:

Example: TM observers O_{TM} observe input-output behavior of systems for single observations and can distinguish two systems iff they compute different functions. SIM observers O_{SIM} observe sequences of input-output pairs, and can distinguish two systems iff they compute different sequences of observations.

Two systems S and S' of a class C are *observation equivalent* for an observer O if for all observations $o \in O$, $o \in B(S)$ iff $o \in B(S')$. This condition is equivalent to $O \cap S = O \cap S'$, which expresses observable behavior as the intersection of behavior generated by systems and behavior observable by the observer. Systems equivalent relative to one observer may be nonequivalent relative to another observer. Systems S, S' are equivalent for observer O if their "projection" onto the subspace determined by observations of O is equivalent. Observation equivalence is parameterized by the class C of observed systems and the class O of possible observations on C. Extending the class of observed systems may require stronger forms of observation, just as extending the class of things to be distinguished may require greater distinguishing power by a human observer.

Observation equivalence: An observer O of a class C of systems determines an observation-equivalence relation E(C, O) such that systems S, S' in C are equivalent iff they are indistinguishable for all $o \in O$. Systems that are not equivalent are *distinguishable* by a finite *distinguishability certificate*.

Algorithmic models deal with the special case C = TM and $O = O_{TM}$, and need not therefore parametrize observation equivalence. Parameterized equivalence allows us to consider a variety of system classes C and observers O. We focus on two classes of systems TM and PTM and two classes of observers O_{TM} and O_{PTM}, where O_{TM} is an observer of input-output pairs and O_{PTM} is an observer of sequences of I/O pairs. For a fixed class C of systems, an observer O_1 is more expressive than O_2 if O_1 has greater distinguishing power than O_2.

Expressiveness: An observer O_1 is *more expressive* than O_2 for the class C if it can make finer observational distinctions: that is, if $E(C, O_1)$ partitions C into strictly finer equivalence classes than $E(C, O_2)$.

We examine systems of the class C_{PTM} of PTMs and show that for this class, observers $O_{PTM}(k+1)$ who can make observations of length k+1 have greater distinguishing power (expressiveness) than observers $O_{PTM}(k)$ restricted to observations of length k [GW].

Proposition: For any k>0, there are pairs of PTMs whose shortest distinguishability certificate has length k+1.

Proof by example: Let M_k k = 1,2, ... be a class of PTMs with bit inputs 0 and 1, and boolean outputs

true and *false*. Let M_k output *true* if the last k values it saw were 1s, and *false* otherwise (if it has not seen that many values, or if some of them were 0s).

We show that $B(M_k)$ and $B(M_{(k+1)})$ accept identical sets of observations of length k or less but have a distinguishability certificate of length k+1. For any input sequence of k or fewer bits with n leading 1s (n >= 0), both $B(M_k)$ and $B(M_{(k+1)})$ admit output sequences consisting of n outputs that are arbitrary followed by k-n values of false. However, the observation consisting of a 0 followed by k 1s yields k outputs of false for M_k and k+1 outputs of false for $M_{(k+1)}$. M_k and $M_{(k+1)}$ therefore have a distinguishability certificate of length k+1 but no distinguishability certificate of length k.

The key idea of this proposition is that observers who can distinguish properties of k+1 length outputs are more expressive than observers who can distinguish only properties of k-length outputs. For example, PTMs that store values of computations in a buffer of length k and then output the buffer at the k+1th input are necessarily distinguishable by k+1 interactions but not by k interactions.

$O_{PTM}(k+1)$ is a strictly more discriminating observer than $O_{PTM}(k)$ for all k, while TMs are in the class $O_{PTM}(1)$. A PTM observer whose memory can handle k PTM observations will always be at a disadvantage over observers whose memory can handle k+1 observations.

The expressiveness of a PTM is given by O_{PTM}, the limit point of approximations $O_{PTM}(k)$ as k -> ∞. Observation equivalences $E(C_{PTM}, O_{PTM})$ are finer than equivalences $E(PTM, O_{TM})$ and finer than equivalences $E(C_{PTM}, O_{PTM(k)})$ for any k. Moreover, $E(C_{PTM}, O_{TM}) = E(C_{PTM}, O_{PTM(1)})$. The observers $O_{PTM(k)}$ determine a hierarchy of progressively more expressive observers such that the least expressive observer with k=1 has the expressiveness of TMs.

The greater power of PTMs is due to the fact that at any step n, they can make use of outputs o_k for k<n in determining their next input. Streams for sequential interaction are expressed as input-output pairs (a_1,o_1), (a_2,o_2), ... where o_k is temporally coupled to a_k and precedes $a_{(k+1)}$. The ability of PTM observers to use outputs o_k for k<n in determining action a_n is expressed by refinement of observation equivalence.

The class of PTMs is larger than that of TMs: TMs are a subclass of history-independent machines reinitialized for each input-output pair, while PTMs have history-dependent behavior for successive actions and can model the behavior of data abstractions such as queues, sequential objects such as bank accounts, and the history-dependent behavior of people.

B(M), the behavior of a machine M defined earlier in this section, only contains finite sequences. However, it can be shown that whenever two SIMS are not equivalent, there will be a finite length distinguishability certificate for them. This is analogous to the proof that inequivalent predicate calculus formulae always have a finite model that serves as a distinguishability certificate. The infinite hierarchy of strictly better approximations via progressively longer finite sequences is analogous to the hierarchy of finite models that approximate an infinite model, or to the hierarchy of approximations to real numbers obtained by lengthening the number of decimal digits.

2.3 Observers and Observed Systems

Actually observed behavior of a system S depends not only on the behavior B_S of the observed system, but also on the behavior B_O observable by the observer O. It is the intersection $B_S \cap B_O$ of system and observable behavior, since behavior must both occur and be observable to be observed. Observers that are TMs can perceive behavior of single I/O pairs but not interactive behavior of I/O sequences. Testing strategies that test for behavior associated with single I/O pairs cannot detect interactive behavior. TM observers cannot perceive the interactive behavior of SIMs, just as blind observers cannot perceive traffic lights or sunsets. Three cases are distinguished depending on whether the behavior B_S of systems S is greater than, equal to, or less than the observation power B_O of observers O that observe them:

$B_S > B_O$: systems have richer behavior than observers can observe

$B_S = B_O$: observation power is matched to system behavior

$B_S < B_O$: observers have excess (unusable) observation power

Systems produce behavior while observers are consumers of behavior. In producer-consumer models excess supply results in products that go unused by the consumer, and excess demand leads to consumer shortages. The products actually consumed (behavior actually observed) is the smaller of supply and demand. For observers, diversity of supplied and demanded products is a better metaphor than volume: the diversity of products purchased at a supermarket is the smaller of supplied and demanded goods. The producer-consumer model is a suggestive model for expressing actually observed behavior of classes of systems and classes of observers.

When a SIM in the role of an observer interacts with a SIM in the role of an observed system, then the following situation obtains:

SIM observers have behavior not expressible by TMs because their actions $a_i = f(o_1, o_2, ...o_{i-1})$ can depend on previous outputs o_k for k<n.

SIM observed systems have behavior not expressible by TMs because their output $o_i = f(a_1, a_2, ...a_{i-1})$ can depend on previous actions a_k for k<n.

The impact of interaction on observers and observed systems is exemplified by question answering. Interactive questioners who can ask follow-up questions have richer behavior than off-line questioners whose questions are determined in advance.

Example (question answering): Interactive answerers can adapt answers to earlier questions, while interactive questioners can adapt to earlier answers. Interactive question-answering supports richer behavior for both questioners and answerers than off-line question-answering for finite sequences of questions. On-line questioning was used effectively in the Starr grand jury to generate large amounts of interactively obtained information that could not have been obtained noninteractively. Greater expressiveness is technically specified by the greater distinguishing power of finer observation equivalence relations. Questioners who can ask k+1 questions have strictly more expressive behavior than questioners restricted to k questions for all k>0. Even the ability to ask a single follow-up question is behaviorally more expressive than noninteractive questioning of TM observers.

The history dependence of observed systems on previous actions (questions) and of observers on previous observations (answers) is summarized in Figure 2.

Observed System: / Observer	TM	SIM
TM	$o_i = f(a_i)$	$o_i = f(a_1, a_2, ...a_{i-1})$
SIM	$a_i = f(o_1, o_2, ...o_{i-1})$	$o_i = f(a_1, a_2, ...a_{i-1})$ $a_i = f(o_1, o_2, ...o_{i-1})$

Figure 2: Interaction Between Observed Systems and Observers

The process of question answering, modeled by the Turing test (section 4.4), is unrealistic when questioners and answerers are modeled by history-independent TMs, and becomes more realistic when questioners are modeled by SIMs that adapt questions to previous answers and answerers are modeled by SIMs that adapt answers to previous questions.

Output is not explicitly specified in the basic SIM definition in the interests of simplicity. It can be modeled as an observable component of the state, identified by a mapping from states to observations.

Partially observable SIM (POSIM): A POSIM has the form POSIM = (S, A, m, O, B), where S, A, m define the underlying SIM, O is a set of observable properties of states, and B: S -> O maps states into their observable behavior. The mapping m of a POSIM is expressed as m: SxA -> SxO or as m: S -> P(AxS)xO.

Example: An automaton whose halting states are observable can be modeled as a POSIM with outputs O = {0,1} and B: S -> O that maps S to 1 for halting states and to 0 otherwise.

POSIMs provide a general model of partial observability of states that specializes nicely to finite automata. The relation between Markov decision processes (MDPs) and partially observable Markov decision processes (POMDPs) is specifiable by the same technique (section 3.2.). Partial observability can be uniformly added to state transition systems by identifying an observable set O of state features and a mapping B: S -> O. Separation of concerns between state-transition and observation behavior is realized by omitting O and B on the assumption that they are implicit and could be made explicit if necessary.

TMs, SIMs and POSIMs model finitely specifiable computing agents with a state-transition structure. TMs have finite sets of states and tape symbols, while SIMs have enumerable finitely specifiable states representable by a persistent working tape and enumerable sets of actions representable as strings. SIMs eliminate the initial state requirement of TMs and express machine behavior in terms of sequences of external actions on a persistent state rather than in terms of a single action on an initial state.

Interaction scales up models of computation from noninteractive state-transition behavior for single interactions to sequences and patterns of interactions that share a history-dependent, persistent, hidden state [We1]. PTMs provide a canonical model for sequential interaction that can be mathematically specified by models that extend inductive to coinductive definition and reasoning.

3. Mathematics of Sequential Interaction

Section 3.1 shows that the mathematical extension from inductive to coinductive models expresses the extension from strings to streams, which is basis of the computational extension from algorithms to interaction. Section 3.2 introduces non-well-founded set theory as a model for streams by replacing the foundation axiom of traditional set theory by the antifoundation axiom. Section 3.3 shows that the extension from algebras to coalgebras parallels the coinductive extension to non-well-founded set theory. Section 3.4 shows that coalgebraic bisimulation expresses on-line observation equivalence more expressive than off-line observation. Section 3.5 examines variants of the Church-Turing thesis for interactive computation.

3.1 From Induction to Coinduction

Induction formalizes the *construction metaphor* of constructing finite structures from primitives, while coinduction formalizes the *observation metaphor* of empirical computer science. The two paradigms play complementary roles in computer science, where the goal of constructing new systems is balanced by the goal of reusing, controlling, and modifying already existing systems. However, the construction paradigm is familiar, while the observation paradigm is unfamiliar, more expressive, and more difficult to formalize.

The observation paradigm expresses the behavior of computing and human agents more completely than TMs by history dependence and persistence. Computer science was concerned almost completely with algorithm construction in its early stages, but is increasingly concerned with specifying external components for reuse and modification as it matures.

We break down induction into primitive constituents to examine its properties and compare it to coinduction. Both induction and coinduction specify sets by one-step iteration rules whose transitive closure determines a fixed point, but they differ in the context of iteration and the nature of the associated fixed point. Inductive definitions specify sets by initiality, iteration, and minimality conditions:

Inductive definition: (1) initiality (base) condition, (2) iteration condition, (3) minimality condition.

Inductive definitions construct new elements of a set from initial and already constructed elements by an iteration rule f(x) -> x. The least fixed point x of the set equations f(x) = x is the transitive closure of the iteration rule. 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 is inductively defined as follows:

Strings: (1) the empty string e∈ A*; (2) if x∈ A* then ax∈ 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 by generating rules (productions) P.

Languages: 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. The derivation relation => is the multistep transitive closure of the single-step derivation relation defined by P.

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

Formal Systems: 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.

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 equivalent expressiveness of TMs, algorithms, computable functions, and formal systems is due largely to their common use of induction to define static domains and dynamic processes of derivation.

Coinductive definitions eliminate the initiality condition of induction, adapt the iteration condition so it becomes a circularity condition, and replace the minimality condition by a maximality condition.

Coinductive definition: iteration condition (circularity condition), maximality condition.

The elimination of initiality corresponds to the extension of *closed systems* to *open systems*. Static initialization is replaced by dynamic incremental late (lazy) binding of interactively supplied information. The switch from minimality to maximality affects the interpretation of the iteration condition. The formal shift from minimal to maximal fixed points captures the intuitive shift from inside-out state-transition specifications that include only elements computable from an initial condition to outside-in behavior specifications that include all behavior not explicitly ruled out.

The distinction between minimal and maximal fixed points is illustrated by the difference in the role of law in a totalitarian society where everything is forbidden that is not allowed and in a democracy where everything is allowed that is not forbidden. The greater resilience of a democracy over totalitarianism is, in a deep sense, the result of feedback and flexibility due to late (lazy) binding of interactive input. Political systems, like computer systems, are more expressive when they permit interactive adaptation than when they are governed by a predetermined set of rules.

Coinductive definitions specify nonenumerable sets of infinite structures by incremental rules for observing them. The iteration condition S -> G(S) determines a greatest fixed point by the equation S = G(S). Eliminating initiality and switching to maximality has a liberating effect on the modeling power of the iteration condition. Rules of inference as well as state transition relations become more powerful when constraints of initiality are removed and maximality replaces minimality.

Streams over an alphabet A can coinductively defined from the inductive definition of strings by removing initiality and replacing minimality by maximality.

Streams are specified by an iteration and maximality condition.

iteration condition: if t is a stream over an alphabet of elements a, and a ∈ A, then (a,t) is a stream over A consisting of an element a followed by a stream t of elements of A.

maximality condition: the set of streams over A is the maximal set satisfying the iteration condition.

Streams are specified by equations whose solutions specify stream elements.

The stream S01: The equations $s_1 = (0, s_2)$, $s_2 = (1, s_1)$ have a solution $s_1 = (0, (1, (0 \ldots)))$ and $s_2 = (1, (0, (1, \ldots)))$. S01 that assigns to each variable an infinitely nested (non-well-founded) stream element.

The stream SB: The equation $s = \{(a, s) \mid a \in A\}$ has as its solution the set of infinitely nested sequences $s = (a_1, (a_2, (a_3, \ldots)))$, where $a_i \in A$, $i>0$. For $A = \{0, 1\}$, the solution is the nonenumerable set of infinite binary sequences.

The stream S01 associates a single infinitely structured stream with each of the variables s_1, s_2. The stream SB associates nonenumerable infinitely structured streams with the variable s, because there are two choices at each interactive iteration step. The relation between streams and SIMs parallels that between strings and TMs. SIMs are stream-processing machines that select among possible streams at each computation step and associate nonenumerable streams of future behaviors with each state.

SIMs: An SIM has the form SIM = (S, A, m), where m: S -> P(AxS) associates with each state s a set of streams $s = \{(a, t) \mid t \in S, a \in A\}$. The future behavior for each state is coinductively defined by choice among streams that directly specify one-step behaviors and indirectly define streams of future behavior.

SIMs determine an *abstraction mapping* from systems into their behavior equivalence class (observation equivalence class). Computation can be viewed as a classification process (abstraction process) that maps systems with unobservable inner structure into behavior equivalence classes.

computation as abstraction mapping: systems (objects, instances) -> behaviors (classes, types)

Coinductive computation classifies nonenumerable classes of objects, like the class of programs with the same behavior, while inductive computation classifies enumerable equivalence classes, like the class of all expressions with the same value.

coinduction supports stronger (nonenumerable) classification than induction

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. 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.

Our earlier analysis of computable functions and TMs showed that they are inductive both in their static definition of domains and in their dynamic definition of computation. It is instructive to examine separately the extension from induction to coinduction at each of these two levels.

Coinductive definition of the domain allows infinite input strings. Coinductive definition of the process of computation allows dynamically generated input by eliminating initiality at each computational step, or equivalently eliminating control structures that uniquely determine the next action. Static coinduction allows streams to be infinite, while dynamic coinduction allows stream elements to be dynamically generated. Static and dynamic coinduction together free streams from both the finite initial input and the unique next action constraints, specifying pure context-independent state-transition relations.

Coinductive static domains increase modeling power from finite to infinite initial inputs.

Coinductive dynamic next actions allow interactive choice at each computation step. The number of possible inputs (worlds) can grow exponentially with the number of steps and becomes nonenumerable for infinite streams.

Static domain coinduction models deterministic TMs with infinite tapes, while dynamic computational

coinduction models *choice machines* whose next action is interactively determined by dynamically-generated input. The term "choice machines", borrowed from Turing's 1936 paper [Tu1], is used here to denote machines that have nondeterministic choice at each computation step but constrain the nondeterminism so it is not completely arbitrary. Choice machines include transducers influenced by feedback from the environment through I/O coupling, and probabilistic machines that model Markov processes.

Both static and dynamic coinduction together are needed to yield the full power of interaction. Dynamic coinduction by itself captures finite interaction of persistent Turing machines. In Figure 3 we call specifications of only static or only dynamic coinduction *half-open systems* and specifications that are both statically and dynamically coinductive *fully open systems*.

	dynamic induction	dynamic coinduction
static induction	Turing machines (closed systems)	choice machines (half-open system)
static coinduction	infinite input TMs (half-open system)	interaction machines (fully open system)

Figure 3: Coinduction and Open Systems

The extension from inductive to coinductive definition and reasoning is a mathematical paradigm shift that extends formal modeling power and the applicability of mathematics to real-world problems. The paradigm shift captures component-based and agent-oriented models of computing technology, but is also significant in a broader interdisciplinary sense because it provides a framework for formalizing a broad range of models of empirical computer science and the natural sciences. A full analysis of broader issues of modeling is clearly beyond the scope of this paper, but we do discuss modeling issues relating to the interactive Turing test (section 4.4), specification (section 3.5), and empirical modeling (section 5.3).

Coinduction provides an interdisciplinary mathematical framework for modeling systems that interact with the external world. Physics deals with observation and control of an already existing world, focusing primarily on empirical models that embody the observation paradigm and only secondarily on the construction paradigm. The tension between construction and observation paradigms arises not only in science and technology, but also in the humanities, where books with titles like "Validity and Interpretation" use the terminology of logical models, and deconstruction of literary texts from the viewpoint of the reader (observer) has led to literary analysis that contrasts sharply with content-based literary analysis.

3.2 Non-Well-Founded Set Theory

Non-well-founded set theory provides a consistent formal model for sequential interaction by formalizing stream behavior, playing a role in interactive models of computation that corresponds to enumerable sets for Church-Turing models. Non-well-founded sets are defined in terms of non-well-founded relations.

Non-well-founded 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.

Non-well-foundedness of relations can be caused by circularity or by infinite chains. The relation $<$ over pairs of integers is non-well-founded because of the infinite chain $i\text{-}1 < i$, while the reachability relation for a directed graph with cycles is non-well-founded because of circularity. The relation R = {(b,a), (c,b), (a,c)} on {a,b,c} has the infinite sequence a,b,c,a,b,... because of circularity.

Non-well-founded sets are defined in terms of non-well-foundedness of the set membership relation $\in$. A sequence $s_{i+1} \in s_i$, i = 0,1,2,... is called a set-membership chain. A set whose members are atomic elements and sets is well-founded if there are no infinite set-membership chains, that is, if every set-membership chain eventually terminates in atomic elements.

Well-founded and non-well-founded sets: A 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.

Non-well-founded set theory can be specified as an extension of Zermelo-Frankel set theory by replacing the inductive *foundation axiom* by the coinductive *anti-foundation axiom* [BM]. Sets in traditional set theory are inductively constructed from atomic elements by union, intersection, complement, and cross-product constructors, which yield sets with atomic elements. The powerset constructor constructs sets that contain sets as elements. 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): Every set is well-founded (ZFC- + FA = ZFC).

Non-well-founded sets with infinite set-membership chains are not constructible inductively by set constructors. They can be coinductively specified as the solutions to systems of set equations.

The set equation x = {a, y}, y = {b, c} has the solution x = {a, {b, c}}, y = {b, c}, obtained by substituting sets for set-valued variables. This is analogous to solving the numerical equations x = y + 3, y = 5 by substituting numbers for numerical variables.

For the recursive set equations x = {a, y}, y = {a, x}, substitution yields a non-terminating (non-well-founded) nested sequence: x -> {a, y} -> {a, {a, x}} -> {a, {a, {a, y}}} -> {a, {a, {a, ... }}}.

Systems of equations that model non-well-founded sets have the following form [BM]:

Flat system of equations: A flat system of 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∪S) is a mapping that specifies for each s∈ S a set of right-hand-side constants in A and variables in S.

The restriction imposed by FA excludes classes of coinductive sets defined by flat systems of equations. Non-well-founded set theory augments ZFC- with the *anti-foundation axiom* (AFA) to obtain ZFA, which admits a larger class of sets and models a larger class of systems than ZFC. The afa has many alternative forms. [BM] specifies AFA by the condition that all flat systems of equations have unique solutions.

AFA: Every flat system of equations has a unique solution (ZFC- + AFA = ZFA)

The AFA asserts not only the existence but also the uniqueness of non-well-founded sets that are solutions to systems of flat equations. Uniqueness implies that sets satisfying the same system of equations are equal (strong extensionality [BM]). Mathematical uniqueness of solutions models physical uniqueness (determinism) of observable behavior of computational and physical systems specifiable by flat equations.

FA specifies precisely the inductively specifiable sets, AFA specifies precisely the coinductively specifiable sets. FA is more restrictive than necessary, and even the AFA may be too restrictive to include all consistent sets. ZFC- specifies a larger class of sets than ZFA that has no well-defined intuitive characterization. It is possible that there are classes of sets between ZFA and ZFC- that provide models for interesting classes of machine behavior associated with stronger definition and reasoning principles than coinduction. Moreover, ZFC- might not be the largest class of consistent sets, and there may be consistent sets with interesting models beyond ZFC-.

The technique of extending domains by assuming the existence of solutions to equations was used to extend the integers to the rationals by assuming solutions of nonsingular simultaneous linear equations with integer coefficients. Later it was used to extend the rationals to irrational and complex numbers by assuming that nonlinear equations have solutions. The AFA is more radical than the extension of integers to rationals, since it extends the universe of sets to include nonenumerable infinite structures. It parallels the mathematical extension of the integers to the reals. The key to the success of both algebraic and set-theoretic domain extensions is the use of coinductive (circular) definition principles.

Flat systems of equations define a coinductive iteration rule m similar to a SIM transition relation (section 2.1), whose solution set m^{∞} is a greatest fixed point and determines a maximality condition. The greatest fixed point determines an observation equivalence relation, called a bisimulation because it models

dynamic equivalence of systems in terms of mutual simulation (section 3.4).

The greatest fixed point determines a denotational semantics for a flat system equations that is a well-founded set. Though solution sets are in principle independent of their construction, the structure of non-well-founded sets strongly suggests an operational semantics by which elements of the solution set are defined and computed. Flat equations determine a computational iteration step for computing solution sets.

Flat systems of equations can model equations whose right-hand sides are ordered pairs. The pair (a, s) is a shorthand for the set P = {a, {a, s}}, which can be expressed as the flat set P = {a, X}, X = {a, s} by introducing an additional variable. In the examples below we freely use this shorthand, which is especially useful in specifying systems of equations for streams.

The mapping function of a flat system of equations can model transitions of a state-transition system. We present two small examples of state-transition systems (Figure 3), followed by a general rule (Figure 3) for modeling state transition systems by flat systems of equations.

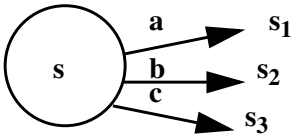
Example (S01): The two-state transition system S01 of Figure 4 is represented by two stream equations $s_1 = (0, s_2)$, $s_2 = (1, s_1)$. The solution of these equations is $s_1 = (0, (1, (0 \ldots)))$ and $s_2 = (1, (0, (1, \ldots)))$. S01 assigns to each variable a non-well-founded set satisfying its set equation.

Example (SB): The equation s = {(0, s), (1, s)} has two recursive choices at each evolution step and determines the nonenumerable set SB of infinite sequences of 0s and 1s.

Figure 4: Labeled Transition System S01 and SB

The systems S01 and SB are instances of *labeled transition systems (LTSs)*.

Labeled transition system: An LTS is a specification of the form LTS = (S, A, m), where m: S -> P(AxS) and s = {(a,s') | there is an outgoing edge from state s labeled a to state s'}. The equation for a state s with three outgoing edges labeled a, b, c to states s_1, s_2, s_3 is shown in Figure 4.



s = {(a, s_1), (b, s_2), (c, s_3)}

Figure 5: Equation for State with Three Outgoing Edges

The solution of a system of equations assigns to each state a non-well-founded set that expresses all possible future behaviors in state s. When steps are interpreted as time steps, behavior is specified for infinite time horizons. LTSs model the state-transition behavior of systems whose mapping (iteration step) is not directly observable. The halting behavior of automata is modeled by adapting the iteration mapping to observe whether the state is a halting state.

Automata: *Automata* can be modeled by a POSIM of the form (S, A, m, O, B) that extend SIMs (S, A, m) with observations O = {0,1} and a mapping B: S -> O such that B(s) = 1 if s is a halting state and 0 otherwise. POSIMs of this form can be modeled by systems of equations of the form m: S = OxP(AxS), where O(s) = 1 if s is a halting state and O(s) = 0 otherwise. The solutions to the equations are non-well-founded sets with additional behavior that allows termination at selected states. Actions a∈ A can be viewed as read-write observations that change the state, while the output function O is a read-only observation that observes a property of the state (the halting property) without changing the state.

Probabilistic transition systems are modeled by an probabilistic iteration mappings.

Markov processes (MPs): MPs have the form MP = (S, A, m), where S and A are state and action sets and m is a probabilistic state transition mapping. MPs can be modeled by probabilistic systems of equations that assign a probability $p_a(s)$ to each action-transition pair (a,s_i), yielding equations of the form s = {$p_a(s)(a,s_i)$ | the transition from s to si along the edge labeled a has probability $p_a(s)$}. Probabilities for a,b,c

in Figure 3 of (.5, .25, .25) yield the probabilistic equation s = {.5(a,s_1), .25(b,s_2), (.25(c,s_3)}. Equations of this form can be expressed as (reduced to) flat systems of equations.

Partially-observable MPs (POMPs): (POMPs) assume that observable behavior is probabilistically determined by the state. They have the form POMP = (S, A, m, O, B), where S, A, m are states, actions, and mappings of an underlying MP, O is a set of possible observations of the MP, and B: S -> O is a probabilistic mapping from states to observations. MPs make probabilistic assumptions about transition structures, while POMPs make additional probabilistic assumptions about observation of state-transition structures, resulting in a two-level probabilistic model. The extension of MPs to POMPs in modeling probabilistic observable behavior by non-well-founded sets parallels the extension of SIMs to POSIMs in modeling automata.

There is an analogy between observation functions O of automata and POMPs. Both determine partial observations of the state. Observers of automata determine a particular property of the state (whether the state halts). POMP observers probabilistically determine whether the system is in a given state.

Decision processes: Decision processes are systems that associate a reward function R with actions and states. They determine policies P: S -> A that map states to actions for systems with directly observable states, and policies P: O -> A for systems whose states are indirectly known through observations. Policies aim to optimize the cumulative reward, yielding rewards with a high expected cumulative value for probabilistic systems. Decision systems for MPs and POMPs (MDPs and POMDPs) are a flourishing area of research in AI that can be formulated in terms of non-well-founded sets.

Now that some examples of non-well-founded sets have been discussed, we reexamine the practical and conceptual motivations for non-well-founded set theory. ZFA extends the class of things that set theory can talk about from inductively specifiable sets of finite objects to coinductive sets of objects that model sequential interaction. Flat systems of equations determine evolution mappings for state-transition systems. The ability to formally model interaction by coinductive extension of set theory is a mathematical breakthrough, extending formal modeling power from Platonic to empirical systems [We3]. A wide range of systems that have historically eluded inductive formalization can be coinductively defined.

Once the antifoundation axiom is accepted as legitimate, the possibility of other coinductive axioms that correspond loosely to other equational extensions of numerical domains suggests itself. Extending systems of equations so that not every equation has a solution is useful in expressing nontermination, while nonlinear equations appear to be useful for multi-agent (distributed) interaction (section 4.2). The set theory analog of the fundamental theorem of algebra (that every algebraic equation has a solution) could well define new kinds of consistent sets whose models express concurrent and distributed interaction.

Most logicians in the 20th century followed Russell in overreacting to paradoxes of set theory [BM] by avoiding axioms like the AFA that support circular reasoning. Though Finsler showed the validity of circular reasoning and discovered non-well-founded set as early as 1925 [Fi], extensions of set theory and logic to circular reasoning were neglected until the 1980s, when Aczel [Ac] showed the relevance of circular reasoning to process models [Mi], and the relative consistency of ZFA (proved by Finsler 60 years earlier) was formalized in a manner acceptable to mainstream logicians. Relative consistency means that ZFA is consistent if ZFC is consistent.

Finsler's approach to set theory was not accepted because he believed in the Platonic reality of pure concepts independently of their verbal or symbolic representation. He believed not only that existence (of a model) implies consistency (of a theory) but also that consistency implies existence. This allowed Finsler to claim existential status for a larger class of entities than formalist logicians, who accorded existential status only to linguistically expressible concepts. Constructive (intuitionist) logicians accord existence to an even smaller class of mathematical objects than formalist logicians.

Finsler distinguished between unsatisfiable circular definitions like Russell's set of all sets that do not contain themselves and satisfiable circular definitions. He accorded existential status to satisfiable (consistent) circular definitions independently of the constructibility or the existence of interesting models. In contrast, constructive logicians excluded circularly defined objects because there was no obvious way of

constructing them, and formalist logicians were skeptical about the existence of models for circularly defined objects because of the paradoxes. The discovery that circularly defined (non-well-founded) sets had interesting computational models caused a resurgence of the study of circular reasoning, and caused some logicians to adopt a broader ontological stance towards questions of existence, reflected in the transition from least to greatest fixed points. Allowing everything that is not forbidden admits a larger class of existential objects than forbidding everything that is not allowed. In particular, greatest fixed points include nonconstructive objects while least fixed points include only constructed objects.

3.3 From Algebras to Coalgebras

The role of coalgebras in interactive computation corresponds to the lambda calculus in Church-Turing models. Algebras model the evaluation of expressions that are inductively defined by an initial algebra and inductively evaluated by reduction processes that specify least fixed points. In contrast, coalgebras model the unfolding of coinductively defined observations that determine a maximal fixed point.

Algebras compute by incremental reduction of operator-operand combinations, using an evaluation (reduction) mapping that maps expressions into a normal form (value). The reduction 3+4*5 -> 3+20 -> 23 has two reduction steps. The lambda calculus and programming languages may have nonterminating reduction processes for which the question of whether expressions have a normal form is undecidable.

Algebras: Algebras are structures A = (S, m: F(S) -> S), where S is 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. The mapping m is decomposable into operations op: S^n -> S of arity n that map operations with n arguments into elements of S.

Algebras determine reduction processes (mappings) from an inductively defined initial algebra of syntactically specified expressions to a quotient algebra that determines the value set.

Initial algebra: For every algebra there is an initial algebra that specifies its domain of expressions and the kind (type) of a class of quotient algebras. An algebra is initial if for an arbitrary algebra Q (of the same kind) there is a unique homomorphism (structure-preserving mapping) from the initial algebra to the quotient algebra Q [Ru1]. The class of all homomorphisms for algebras of a given kind form a category whose identity element is the initial algebra.

The extension from algebras to coalgebras is an algebraic analog of the extension from induction to coinduction and from well-founded to non-well-founded sets. Coalgebras determine reduction processes (mappings) of systems into observation equivalence classes. They model the process of observing system behavior by performing actions on or observations of state-transition systems with an unobservable system state. The algebraic iteration step F(S) -> S of incremental expression evaluation becomes the coalgebraic iteration step S -> Γ(S) of system evolution, which incrementally maps states into behaviors. The context of iteratiuon is broadened to eliminate initiality and permit external control (binding) of iteration steps so that seqeuences of coalgebraic steps are represented by streams rather than strings.

Coalgebras: Coalgebras are structures CA = (S, m: S -> Γ(S)), where S is a set of states and m: S -> Γ(S) is a behavior-preserving homomorphism of observed systems with an unknown state S into unfolded set of behaviors (observations).

The evolution mapping can sometimes, though not always, be decomposed into state-transition (read-write) operations and observation (read-only) operations. Coalgebras whose evolution mappings are one-to-one are called final coalgebras. States of final coalgebras for state-transition systems represent unfolded system behaviors.

Final coalgebra: For every coalgebra there is a final coalgebra that uniquely specifies its behavior and determines the kind (type) of a class of coalgebras that map into the given final coalgebra. A coalgebra is final or terminal if for an arbitrary coalgebra of the same kind there is a unique final coalgebraic mapping into the final coalgebra [Ru1]. The final coalgebra of a system is a normal form for a (nonenumerable) equivalence class of systems having that behavior. The class of homomorphisms for coalgebras of the

given kind is a cocategory whose identity is the final coalgebra.

Coalgebraic mapping functions for labeled transition systems are specified by flat systems of equations and are mapped into final coalgebras that are non-well-founded sets.

Coalgebra for LTS: An LTS with state set S can be specified by a coalgebra (S, m: S -> P(AxS)). The evolution mapping m is specified by a system of equations that for each state s specifies the set of pairs (a,s) corresponding to outgoing labeled edges of the LTS. The set of all possible unfolded behaviors of an LTS is a final coalgebra whose set S is the solution set of the equations. Its mapping function is one-to-one.

Coalgebras for automata have both a state-transition and an output operation. Output operations observe the system state without disturbing it and are read-only.

Coalgebra for automata: A coalgebra for automata has the form AU = (S, m: S -> OxP(AxS)). Its evolution mapping admits output as well as state transitions of an LTS.

Final coalgebras have no proper quotient algebras (that is part of the meaning of final), but may have distinct subalgebras, corresponding to behaviors associated with distinct systems. We present a particular final coalgebra FCA for a broad class of languages and automata, taken from [Ru2].

Final coalgebra FCA: The final coalgebra FCA for the class $\mathcal{L} = P(A^*)$ of all languages over an alphabet A has a nonenumerable set $P(A^*)$ of nodes. For every state L and $a \in A$, there is a transition L -> aL_a where $L_a = \{v \mid av \in L\}$, called the a-derivative of L, is obtained by selecting all strings of L starting with a and chopping off the a. States containing the empty symbol e are final states that can be observed by an operation o: L -> {0,1}, where o(L) = 1 if L contains e and 0 otherwise. The node L represents the behavior of the language L in that the set of possible observations from any $L \in \mathcal{L}$ is precisely the language L itself.

The node corresponding to L = {a, ab,bc} has outgoing transitions labeled a to L_a = {e, b} and labeled b to L_b = {c}. Since L_a contains e, $o(L_a) = 1$, while $o(L_b) = 0$. Each node L is the root of a subalgebra expressing the behavior L. The subalgebras of FCA are in one-to-one correspondence with its nodes.

Each state L of FCA determines a distinct subalgebra, where L's reachable states represent the future behavior of each state. Final coalgebras in general have the property that each state represents the future behavior of the system for that state.

The relation between algebraic evaluation mappings and coalgebraic evolution mappings is illustrated in Figure 6. Algebraic mappings are structure-consuming, mapping expressions into values, while coalgebraic mappings are structure-producing, mapping systems with unknown structure into behaviors.

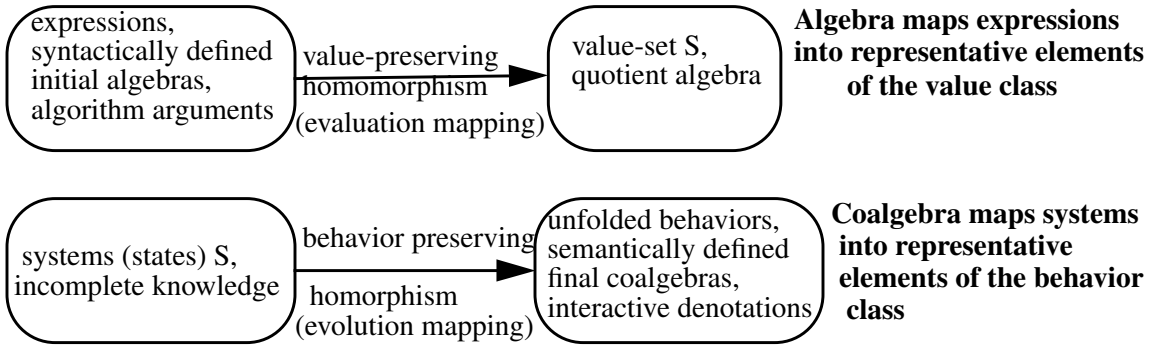


Figure 6: Algebraic Evaluation and Coalgebraic Evolution

Both algebraic and coalgebraic mappings are property preserving mappings that implement an abstraction process. They identify elements with the same value of a property P while distinguishing between elements with different values of a property P. However, the identification of values of expressions is both

substantively and qualitatively different from the identification of systems with the same behavior.

Coalgebraic abstraction that identifies nonenumerable systems into behavior-equivalence classes is stronger than algebraic abstraction that maps enumerable expressions into value-equivalence classes. For example, equivalence classes of programs with the same value are known to be nonenumerable. The process of identifying algebraic expressions with the same value expresses algorithmic computing, while the process of identifying systems with the same behavior expresses interactive computing (Figure 7).

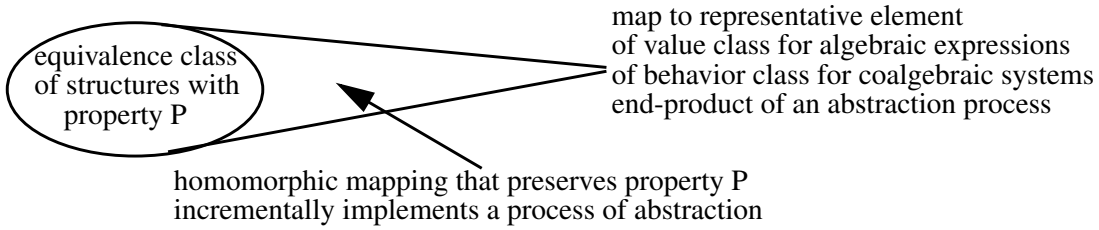


Figure 7: Computation as a Process of Abstraction

Coalgebras support a variety of forms of evolution mapping that determine different models of computation. Non-well-founded set theory is modeled by coalgebras whose evolution mappings m: S -> Γ(S) specify flat systems of equations. However, coalgebraic mappings are not restricted to be of this form. For example, m can be a partial mapping such that not every equation has a solution. Systems whose observation can cause sequences of silent moves have been studied in [VW, Mi]. Systems that admit nonterminating sequences of silent moves, like sequential objects of object-oriented programming, are not specifiable by equations with a unique solution that satisfies the AFA. Nontermination is denoted by the symbol ⊥ and can be specified by coalgebras with mappings m: S -> Γ(A, S) that are partial functions over AxS.

Partial coalgebra: A partial coalgebra P = (S, m: S -> Γ(A, S)) has a partial mapping function m undefined for some elements s ∈ S, and a∈ A, corresponding to nonterminating actions a in state s.

Nondeterministic coalgebras have evolution mappings m: S -> Γ(A,S) that may associate more than one possible next state with some action-state pairs (a,s) of AxS.

Nondeterministic coalgebra: A coalgebra ND = (S, m: S -> Γ(A,S)) whose mapping function m maps s into a set of next states for actions a∈ A and states s∈ S is a nondeterministic coalgebra.

Partial and nondeterministic coalgebras have evolution processes that do not satisfy the AFA. However, since systems with the given behavior exist, axioms that embody this behavior are consistent.

Conjecture: Coinductive axioms ZFC++ that extend ZFC- to model realizable system behavior are consistent if ZFC is consistent.

Proof: Systems that realize the axiomatic behavior are a model for the axioms.

Evolution mappings of traditional coalgebras have a simple model of time that marches forward in discrete uniform intervals. All actions a∈ A are assumed to take precisely one time step. The stationary mapping function m determines a final coalgebra $M = m^{\infty}$ by a time-independent sequence of mappings m. M is temporally decomposable into discrete steps governed by a time-invariant rule parameterized by actions.

In section 4.2 we consider multi-agent systems that require coordination of multiple evolution mappings m: S -> Γ(A, S). For multi-agent systems, actions a become transactions that may take more than one time step, and mappings m may depend on uncompleted actions and are not time-independent. Evolution mappings have the general form m: F(A, S) -> Γ(A, S), since each evolution step depends on incomplete actions as well as the current state. Serializability of transactions implies the existence of an equivalent mapping m_t: S -> Γ(A, S) where the granularity of m_t is determined by the granularity of transactions.

Multi-agent modeling requirements require a variety of extensions of the form of evolution mappings.

Mappings m_t: S -> Γ(A, S), where A are transactions that span multiple time intervals, correspond to an extension of dynamical systems from Markov to non-Markov processes that view time as an active rather than passive variable in specifying system evolution. The dependence of the mapping on uncontrollable inputs of other streams introduces both nonlinearity of nondeterminism (section 4.2).

Multi-agent behavior cannot be unfolded by iteration of a stationary mapping m such that $m^{\infty} = M$. Temporal decomposition of behavior is not in general possible either for multi-agent computers or for histories of a distributed interconnected world. though time progresses linearly, multiagent (distributed) behavior cannot be linearly described).

Nondecomposition of behaviors into mappings for discrete time steps corresponds loosely to nonlinearity and could in principle be specified by set-theoretic axioms that specify solutions of nonlinear equations. The relation between the fundamental theorem of algebra (that every algebraic equation has a solution) and models of multi-agent computation is intriguing.

Exploration of more complex axioms for set theory that model multi-agent coalgebras, and determination of consistency for such models, is beyond the scope of this paper. Consistent extended set theories should exist for axioms that model interactive systems that can actually be built. Extensions of non-well-founded set theory similar to those above are touched on in the final chapter of [BM].

3.4 Bisimulation

Equivalence is a subtle concept that may be progressively specialized from equality (of all properties) to similarity (equivalence of some properties) and simulation (dynamic equivalence of behavior). Symmetry of equivalence gives rise to bisimilarity and bisimulation that capture two-way step-by-step simulation of processes. Bisimulation captures mutual two-way dynamic behavior simulation between two systems, and is the natural extension of static equivalence to dynamic sequential interaction. Bisimulation is a coinductive equivalence relation between non-well-founded sets that models the behavioral equivalence of streams. The mathematical question "when do two equations have the same solution?" models the computational question "when do two systems have the same behavior?". For coalgebras, this question becomes "when do two coalgebras have the same final coalgebra?"

Equivalence for sets is specified by the *principle of extensionality.* Two sets S, T are equal (S = T) if:

a) for every $s \in S$ there is a $t \in T$ such that $s = t$

b) for every $t \in T$ there is an $s \in S$ such that $s = t$

Equality of sets is recursively defined in terms of equality of subsets down to an arbitrary recursive level. This recursion always terminates for well-founded sets, giving us an inductive approach to proving set equality. For non-well-founded sets, extensionality yields a circular, coinductive form of extensionality called strong extensionality [BM] that transcends inductive extensionality of finite structures. Strong extensionality of non-well-founded sets determines equivalence of infinite structures by interactive dynamic simulation processes.

Two sets S, T are equivalent if there exists a *bisimilarity relation* $R \subseteq SxT$ involving all members of S and T; R is recursively defined as follows:

for all $s \in S$ and $t \in T$, R(s,t) iff s and t are atomic and s=t, or

a) for every $s' \in s$ there is a $t' \in t$ such that R(s',t')

b) for every $t' \in t$ there is an $s' \in s$ such that R(s',t')

The primary difference between bisimilarity and the earlier definition of equivalence is the replacement of extensional (inductively defined) equality "=" by a coinductively defined relation R. More than one bisimilarity $R \subseteq SxT$ may exist for a given pair of sets S,T. However, the union of all such R is unique, and is the greatest bisimilarity. When bisimilarity is interpreted as equivalence of system behavior for all states $s \in S$ and $t \in T$, the greatest bisimilarity includes all pairs of states that preserve behavior for every possible action $a \in A$. This greatest bisimilarity expresses coinductive maximality and specifies coinductive equiva-

lence for non-well-founded sets and the systems that they model.

When S and T are state sets of systems and R(s, t) means that s and t have equivalent behavior, then bisimilarity expresses simulation of each system by the other, and bisimilarity of sets becomes bisimulation of systems. Bisimulation relations R model mutual on-line simulation of sequences of actions in one system by sequences of actions in the other. Bisimulation of systems is a specialized form of bisimilarity for behavior equivalence between evolving systems about which we have incomplete knowledge.

Bisimulation for coalgebras is defined by mutual simulation of their system evolution functions.

Bisimulation of coalgebras: Two coalgebras CS = (S, m:S->Γ(S)) and CT = (T, m':T->Γ(T)) are related by a bisimulation relation $R \subseteq SxT$ if for each $s \in S$ and each evolution step of CS there is a $t \in T$ and evolution step of CT that preserves R, and conversely.

For automata, this definition specializes to mappings Γ of the form O x P(AxS), for output O = {0, 1} and an action set A. That is to say, each state s is mapped to an output value and a set of pairs of the form (a, s) where $a \in A$, $s \in S$.

Bisimulation of automata: Two automata AS = (S, m:S -> OxP(AxS)), AT = (T, m': T -> OxP(AxT)) are bisimilar if there exists a relation $R \subseteq SxT$ involving all members of S and T such that:

for all $s \in S$ and $t \in T$, R(s,t) iff o(s) = o(t) and

a) for every pair $(a, s') \in \Gamma(s)$, there is a pair $(a, t') \in \Gamma(T)$ such that R(s',t')

b) for every pair $(a, t') \in \Gamma(T)$, there is a pair $(a, s') \in \Gamma(S)$ such that R(s',t')

When systems are rooted, bisimilarity of roots may serve as a "starting point" for constructing the bisimilarity relation, which allows an alternate definition of bisimilarity of LTSs, one based on the least fixed point approach. Labeled transition systems (LTSs) are rooted automata without an output function, that is to say, each system has a distinguished state ("root"), and the roots must be bisimilar. Two LTSs S and T are bisimilar if the bisimilarity relation between their states includes (root(S), root(T)).

Both the rooted and unrooted approaches yield the same bisimulation relation between rooted systems; if a bisimilarity relation exists between all their states, then both the least-fixed-point and the greatest-fixed-point approach will find some such relation. However, the least-fixed-point approach cannot apply to unrooted systems.

Bisimulation equivalence can be proved for finite automata (this follows from the Myhill-Nerode theorem) but not for SIMs whose number of states may grow arbitrarily large. Equivalence cannot be proved for physical or computing systems with an unknown inner structure, but it can be falsified at any time by finding observations that cause differences of behavior. This is the basis for assertions by Popper [Po] that scientific theories can only be falsified but never verified and by Dijkstra [Di] that testing can discover incorrectness but never prove correctness of programs.

3.5 Church-Turing Thesis

The Church-Turing thesis may be specified as follows:

Church-Turing thesis: The intuitive notion of effective computability of functions (over positive integers) corresponds to formal effective computability by the lambda calculus (Church) or TMs (Turing).

This thesis has the form “the formally definable notion X corresponds to an intuitive notion Y”. Theses that relate formal to intuitive concepts are motivated by two kinds of questions:

1. For a given formal concept, what intuitive notion does it formalize?
2. For a given intuitive concept, what is its appropriate formalization?

Church’s thesis was motivated primarily by the first question, aiming to provide intuitive underpinnings for already developed formal concepts. Church related the robustness of computable functions for alternative formal models to a natural, independent, intuitive notion of function computation. He started from the remarkable fact that the lambda calculus, TMs, and recursive function theory are equally expressive, and

identified the intuitive notion corresponding to this robust formal notion as the effectively computable functions [Ga]. Church's thesis links the formal abstraction of computable functions as defined by TMs and the lambda calculus to an intuitive notion of functional computation.

However, Church's intuitive notion of computation is, not unexpectedly, dependent on and limited by his formal model. When the intuitive notion is broadened to include interactive computation, Church's formal notion must be correspondingly broadened. A Church-style thesis for sequential computing can be formulated as follows:

Church-style thesis for sequential interaction: The intuitive notion of effective computability for sequential interaction corresponds to specifiability by non-well-founded sets, and to computability by SIMs or PTMs.

SIMs can perform a larger class of tasks (problems) than TMs. Does the larger class of tasks (problems) specifiable by SIMs include noncomputable functions of positive integers, or are non-well-founded sets not functions but a nonfunctional class of computable things called computable nonfunctions?

Question: Are transductions from streams to streams a) computable functions from integers to integers, b) noncomputable functions from integers to integers, or c) functions over coinductive domains that cannot be expressed as domains of integers?

Answering these questions requires a careful analysis of what it means to be a function. In restricting himself to functions over positive integers, Church considered functions over inductively-defined domains and appears to have excluded functions over coinductively-defined streams. The question "what functions are intuitively computable?" can be refined into the following two questions:

1. Is the intuitive notion of computable functions for inductively defined domains captured by TMs?
2. Are computations for domains that are not inductively definable intuitively computable?

When stated in this way it becomes clear that Church limited himself to functions over inductively definable domains, not only in his definition of "computable function" but also in his notion of "noncomputable function". Computable functions can be defined as functions whose computations as well as domains are inductively definable, while functions with coinductively definable computation processes over inductively definable domains with can be noncomputable in the Turing sense. When X, Y are non-well-founded sets, then the mapping m: stream -> Y is not a Church-Turing function from integers to integers, though it is a function in a broader sense.

Church-Turing functions have the property that elements of their domain are completely determined before the computation starts. Though inputs could be incrementally supplied, such incremental inputs must be equivalent to a predetermined input. Transductions from streams to streams are not equivalent to mappings from predetermined domain elements to a range and are therefore not Church-Turing functions. Moreover, mappings that can be described functionally after they are completed (because they are then completely determined) do not qualify unless they are completely determined prior to the computation. Retrospective functional description of a sequence of events does not imply that they could have been described functionally in advance. Functions must be describable as rules of correspondence between a predetermined domain and a range before they are actually computed.

Transductions F: stream -> stream are not Church-Turing functions from integers to integers. Dynamic binding of elements of the stream to information not available prior to the computation, such as outputs of the interaction machine, contradicts the requirement of a mapping from predetermined inputs to outputs. When transductions of streams to streams are included in the intuitive notion of what computers compute, the intuitive notion of function can no longer be identified with functions from integers to integers.

Church-style theses that narrow the scope of X and Y have been considered.

Strong (Physical) Church's Thesis: Any physical computing device can be simulated by a TM in a number of steps polynomial in the resources used by the computing device.

The physical Church thesis appears pragmatically true for machines that obey the laws of classical physics, though harnessing turbulence or chaos could possibly violate the physical Church thesis. However, Feynman [Fe] conjectured that quantum computers could not be simulated by TMs in polynomial time. There is strong evidence that quantum Turing machines (QTMs) [De, BV] are a counterexample to the physical Church thesis because they allow functions such as prime factorization for which there are no known polynomial algorithms to be computed in polynomial time [Sch]. Their ability to update a superposition of configurations at each state transition can in principle be harnessed to perform an exponential amount of work in polynomial time. The idea that state transitions of natural devices can exhaustively explore an exponential space of possibilities in polynomial time is plausible, though only a polynomial subset of such spaces can be explored: the trick is to organize the computation so that the computational device explores the space for the particular problem.

QTMs are defined in [BV] so they can be observed only once. Interactive QTMs (IQTMs) that model sequences of QTM observations would be more expressive than QTMs, just as IMs are more expressive than TMs. IQTMs do not appear to satisfy the modified Church thesis because they can handle exponentially growing classes of interactive alternatives. The hidden-interface model can model QTMs and IQTMs without requiring nondeterministic or probabilistic behavior.

Whereas the physical Church-style thesis considers X, Y weaker than Church, the interactive Church-style thesis considers X, Y stronger than Church. IMs express interactive computations that are not computable by TMs, where expressiveness is defined in terms of the ability to make observational distinctions. Sequential interactive computing provides a stronger intuitive notion of computing than Church's thesis, while nonsequential interactive models provide a still stronger notion as shown in the next section.

4. Beyond Sequential Computation

Section 4.1 develops a model of distributed information flow [BS] in terms of a mathematical model of interfaces and of infomorphisms that relate unobserved behavior at remote interfaces to behavior at directly observable interfaces. Section 4.2 shows that multi-agent interaction machines (MIMs) that express patterns of interaction of autonomous agents are more expressive than SIMs. Section 4.3 connects nondeterminism of observers of MIMs to quantum-theoretic nondeterminism and proposes a hidden-interface model that extends Einstein's hidden-variable model as an explanation of physical nondeterminism. Section 4.4 extends the Turing test to behaviorist models of sequential and distributed "thinking", showing that finite agents with greater extensional behavior also have greater intentionality.

4.1 Distributed Information Flow

A distributed system may be defined as a system with geographically or logically separated interfaces. One of the primary roles of distributed systems is that of facilitating *distributed information flow* among external agents that observe or access system interfaces. Systems transmit observed data by unobservable information flow among system components. Systems can be viewed as *information channels* [BS] that *carry information* about behavior at remote interfaces to interfaces at which behavior is directly observed. Regularities of system behavior allow observers to make inferences from observed behavior at directly observable interfaces about unobserved behavior at remote or inaccessible interfaces.

Examples: The observation interface (control panel) of an airplane carries information about the engine by homomorphically expressing the engine's unobserved behavior through regularities of the system that connects the engine to the interface (Figure 8). The observation interface of a nuclear reactor carries information about the unobservable behavior of its radioactive core. Systems act as two-way

information channels that support observation and control of remote behavior by interface actions.

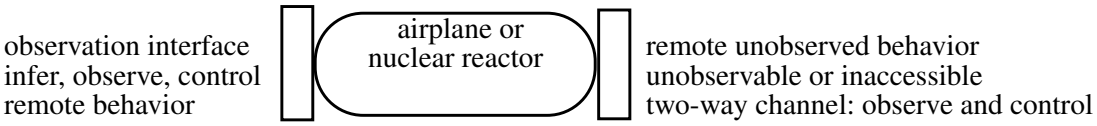


Figure 8: System as Information Channel that Carries Information about Unobserved Behavior

Systems act as information carriers that allow external agents to make inferences about remote behavior from control panel observations and to control remote behavior by control panel operations. Inferences about remote behavior flow in the opposite direction from information flow of the system. This analysis of distributed information flow for single events is noninteractive, but provides a foundation for models of interactive distributed behavior in section 4.2.

Figure 9 shows the role of interfaces in mediating between the system that carries information and its externally observable behavior. From the viewpoint of observers, interfaces display inner states of particular systems as external screen objects that represent classes of inner states. Interfaces correspond to assertions of the form "a is A", where a is an inner system state and A is an external (displayed) event type:

interface: maps unobservable particular system state into an observable event type

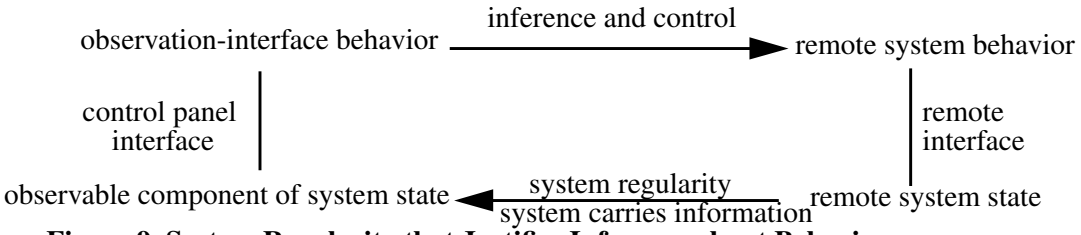


Figure 9. System Regularity that Justifies Inference about Behavior

The information carrying system need not interpret or understand the information that it carries as long as it faithfully transmits information among its interfaces. Each interface mapping transforms inner state information to an external representation that may be interpreted by external computers or human agents.

Information flow across interfaces transforms a particular instance of a system state to an event type. Control panel displays carry information about particular behavior of a specific system but are interpreted externally as type of event. Mappings at interfaces can be viewed as classifications that map particular system events into event types of external observation equivalence classes. Mappings from instances to classes can be viewed as classifications [BS].

A *classification* $\mathbf{A} = (A, \Sigma_A, |=_A)$ consists of a set A of objects or tokens, a set Σ_A of classes of tokens, and a binary relation $|=_A$ that relates tokens to classes.

Examples: Function evaluation maps arguments into value classes that do not distinguish between domain elements that map into the same values. Observers of systems perceive observation equivalence classes of behavior rather than actual system behavior.

Classifications model the observation of system behavior and reflect the fact that observers perceive only the observation equivalence classes to which objects belong and not the objects themselves. Classifications are a form of Chu spaces, whose properties have been extensively studied by Pratt and others [Pr]. Classifications are a specialization of Chu spaces to a particular class of interpretations, where $|=$ is a relation between particular objects that carry information and observation equivalence classes of event types that they induce.

User inference about remote behavior from directly observable behavior is justified by the notion of an infomorphism [BS] between classifications (figure 10). Infomorphisms specify correctness conditions for inference about remote behavior. They also determine correctness conditions for interpreters that map user specifications $\alpha \in \Sigma_A$ into system specifications Σ_C executable by a system C.

An infomorphism between two classifications $\mathbf{A} = (A, \Sigma_A, |=_A)$ and $\mathbf{C} = (C, \Sigma_C, |=_C)$ is a pair of mappings f_{lower}:C -> A and f_{upper}:Σ_A -> Σ_C such that for all $c \in C$ and $\alpha \in \Sigma_A$ the following condition obtains:

$f_{lower}(c) |=_A \alpha$ iff $c |=_C f_{upper}(\alpha)$

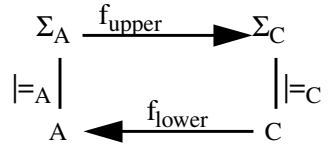


Figure 10: Informorphism Between Two Classifications (Abstraction of Figure 4)

The interpretation (inference) $f_{upper}(\alpha) \in \Sigma_C$ of directly observed behavior $\alpha \in \Sigma_A$ obtained by mapping $c \in \Sigma_C$ by f_{lower} and |=A is always the same as the mapping of c by |=C into Σ_C

The lower mapping captures the causal mapping of remote systems to directly observable subsystems, and the upper mapping captures the inference relation from behaviors (classifications) of subsystems to behaviors of remote systems. Infomorphisms express relations between two classifications of the form “a is A” iff “b is B”, so that inference about behavior is valid for both observation and control. Commuting of the diagram expresses the fact that inferences are always valid. In the terminology of Chu spaces, this fundamental property of infomorphisms is an instance of a Chu transform.

Infomorphisms model the process by which systems deliver information about unobserved behaviors to observable interfaces by means of inner shared state-transition behavior. Though the inner system structure of airplanes and nuclear reactors cannot be observed, regularities of unobservable inner behavior allow systems to function as channels that carry information.

Infomorphisms allow users to infer unobserved remote events from directly observed interface events. in Figure 10 we can distinguish three forms of causal relations:

system-system causality: f_{lower} causally relates system behavior at C to that at A

system-behavior causality: interfaces causally relate system states to observable event types

behavior-behavior causality: f_{upper} is a causal relation between events induced by system behavior

Infomorphisms allow users to infer causality among external events from system-system causality of system regularities and system-behavior causality of interfaces. Infomorphisms express highly constrained forms of distributed behavior. In the next section we consider a less constrained from of distributed behavior associated with autonomous agents whose nondeterministic observation and control problems that differ from those of control systems designed to provide users with control over remote behavior.

Infomorphisms between a controlled system and one or more control panels (interfaces) consider the remote impact of a single noninteractive step and may be modeled algorithmically. Distribution is orthogonal to interaction in that noninteractive distributed systems can be algorithmic while interactive nondistributed systems are not [We1]. The analysis of interactive system behavior for patterns of autonomous, interactively concurrent, events requires the infomorphism model of [BS] to be extended.

4.2 Multiple-Stream (Distributed) Interaction Machines (MIMs)

Distributed systems with autonomous agents are modeled by multi-stream IMs (MIMs). SIM interaction has the property that an observer (stream) together with the observed system is closed, while autonomous (multi-stream) MIMs cannot be closed by composition with any single agent because of autonomous interaction with other agents.

Multi-stream IM: A *MIM* is a state-transition system that interacts with autonomous multiple streams.

Examples of MIMs: airline reservation systems, the World-Wide Web, distributed databases, multi-agent games, multi-agent negotiation.

This definition is esthetically appealing because it defines multi-agent systems in terms of a notion of interactive concurrency that is the interactive counterpart of algorithmic concurrency. Algorithmic concur-

rency is defined by concurrency of process execution, while interactive concurrency is defined by concurrent interaction of multiple agents (second-order concurrency of streams). There is a strong connection between autonomous agents and interactive concurrency: being autonomous can be defined as being interactively concurrent. We will see below that the difference between MIMs and SIMs is related to the difference between interleaving concurrency and true concurrency: interleaving concurrency can be expressed by SIM behavior while true concurrency cannot.

MIMs are distributed systems in the sense of section 4.1, expressing the behavior of finite single systems with multiple related (but autonomous) interfaces. In showing that MIMs are more expressive than SIMs we extend the expressiveness of finite agents beyond sequential interaction, capturing higher forms of behavior of finite agents such as coordination and collaboration, and showing that collaboration is strictly more expressive than sequential interaction. MIMs that focus on behavior of a single finite system in a multi-agent environment capture some important forms of behavior:

1. MIMs provide a behavior model for single agents dealing with multiple autonomous agents and a framework for specifying the expressiveness of such agents.

2. MIMs model the observable behavior of physical systems and provide a framework for the analysis of physical models of observability like quantum theory.

3. MIMs provide a framework for the extension of behavioral models like the Turing test to multi-agent behavior (section 4.4).

The proof is based on the fact that input-output actions of streams are transactions, and that transactions of multiple autonomous streams cannot be serialized by a single stream. MIMs support the behavior of nonserializable transactions, which are a form of true concurrency [Pr]. The result that MIMs are not expressible by SIMs is related to the nonexpressibility of true concurrency by interleaving.

Proposition (expressiveness): MIM behavior is not expressible by (reducible to) SIMs.

Proof: Each streams for a MIM is a sequence with I/O coupling of the form (i_1, o_1), (i_2, o_2), ..., where o_k follows i_k and precedes i_{k+1} for $k>0$. Consider a MIM that interacts with a user who asks a question and delegates part of the task of answering to an expert. Let (iu, ou) be the I/O interaction with the user and (ie, oe) be the delegated interaction with the expert. Since ie depends on iu (the expert is consulted about a user request) and ou depends on oe (the expert's reply is used to answer the user request) there is no way to merge the two streams while preserving input-output coupling. The two interactions, considered as transactions, cannot be serialized, because of the circular dependency.

This proof shows that finite agents that interact with both an input stream and a delegation stream cannot be expressed by SIMs because the input and delegation streams cannot be merged while preserving I/O coupling. The intuition that delegation cannot be modeled by SIMs is formally expressed by impossibility of preserving I/O coupling. This problem is related to merging of autonomus streams of transactions or actions with duration.

Multiple input streams can be merged into a single stream when there is no I/O coupling. A merge program for two interactive streams that synchronously compares their first elements and removes the smaller one is a sequential interactive process. Merging is a prototypical reduction procedure for reducing multiple streams to a single stream. Actor systems that merge inputs asynchronously into a single input stream without coupling them to outputs achieve reduction of inputs from multiple sources into a single stream. Irreducibility occurs when atomicity requirements of multiple streams cannot be preserved under merging.

The condition that the transactional integrity of multiple streams can be preserved under merging corresponds to serializability.

Proposition: Multiple streams of transactions are expressible by an SIM iff they are serializable.

Proof: Serializability of transactions of multiple streams means that for every actual execution there is an equivalent sequential order of transaction execution. Thus there is a SIM whose behavior is equivalent to every actual execution of the MIM. Conversely, if the transactions are not serializable, there is no single-

stream arrangement of transactions whose execution is equivalent to the multiple-stream execution.

Multi-stream behavior can in many cases be serialized, but important higher forms of behavior, such as coordination of conflicting tasks, is nonserializable. Handling of conflicting autonomous demands by CEOs and senior managers is performed more efficiently in a nonserializable mode even when it can be serialized. Serializability radically reduces the range of possible MIM behaviors to those that can be modeled by sequential interaction. High achievers are distinguished by the ability to deal efficiently with multiple streams while average achievers perform algorithmic or sequentially interactive tasks. Nonserializable behavior is more common in nature and among human agents than is generally realized.

Though multi-agent expressiveness cannot be easily modeled, it is a ubiquitous form of system behavior that parallels human behavior in coping with asynchronous demands. Computational behavior may be classified by whether it is algorithmic or interactive and interactive behavior may in turn be classified by whether it is sequential or multi-agent (Figure 11).

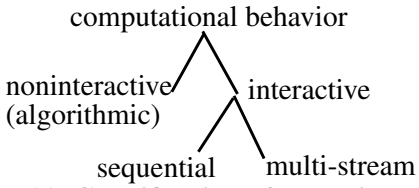


Figure 11: Classification of Behavior by Expressiveness

Multi-stream systems can in principle be described by extended coalgebras of the form MIM = (S, M: F(A, S) -> Γ(A, S)), where M maps a configuration of states and partially completed transactions to a new configuration and partially completed transactions. If the actions A are serializable transactions, some degree of simplification is possible, though temporal granularity is determined by transactions rather than a user-defined clock. People with interpersonal skills may in fact perform a mapping that approximates serialization of transactions in organizing a collection of tasks, though tasks are often more efficiently performed by allowing some degree of nonserial behavior.

Development of coalgebraic models of computation that correspond to actual forms of multi-agent behavior is beyond the scope of this paper. But the fact that multi-agent behavior can in principle be specified by extended coalgebras is useful in understanding the nature of multi-agent models of computation.

4.3 Nondeterminism as Incomplete Observability

A MIM is a tool for studying the interaction of k+1 entities, where one plays a special role as an observed system and the remaining entities are observers, modeled by streams. For interacting SIMs, k = 1 and the observer with the observed system form a closed system. When the observed system is a MIM, k>1 and a primary observer representing the experimenter is distinguished from k-1 secondary observers. MIMs appear nondeterministic from the viewpoint of a primary observer who cannot predict or control the effects of secondary observers and may be unaware of their presence. Primary observers perceive MIMs as subjectively nondeterministic even though, viewed by an omniscient multi-agent observer (God), they are objectively deterministic (Figure 12).

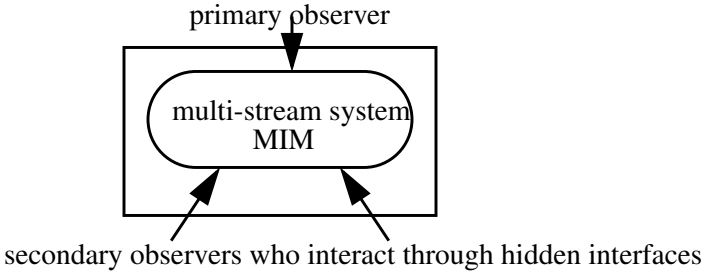


Figure 12: Multi-Stream Interaction of Primary and Secondary Observers

We conjecture that the nondeterminism of primary observers of a MIM is related to the physical mechanism for nondeterminism of quantum theory. Though Figure 12 does not capture the physics of quantum theory, the inability of primary observers to provide a complete and therefore deterministic account of observed-system behavior could potentially provide an explanation of quantum nondeterminism. If this explanation is correct, then nondeterminism is due to limitations of sequential observers in observing distributed systems rather than to inherent nondeterminism in nature. Human observers, as well as computers, observe distributed systems through specific observation interfaces and are incapable of observing complete system behavior.

Experiments of classical physics and quantum theory generally assume that observed systems can be isolated and their interaction can be completely controlled by the experimenter. In fact, physical systems are connected in multiple ways to their environment, for example through electromagnetic and gravitational forces. Random noise is an example of an effect by secondary observers that may be significant at the quantum level though ignorable at the classical level. Quantum observations have an "equal and opposite" effect on observed systems (Newton's first law applies), while classical observations do not affect observed systems (Newton's first law is ignored) [We3].

Quantum theorists like Bohr believed that the observed nondeterminism of quantum theory reflected unexplainable inherent nondeterminism of the real world, while Einstein believed that "God does not play dice" and proposed a hidden-variable model that aimed to explain the observed nondeterminism of quantum theory. Alain Aspect, in an experiment in the 1980s based on Bell's inequalities [Per] showed experimentally that hidden-variable models could not explain quantum behavior. However, the inference that physical laws are inherently nondeterministic, and that Einstein's sense of esthetics may have clouded his technical judgment, is unwarranted, since MIMs can provide a model for reconciling the observed nondeterminism of quantum theory with objective determinism [We3].

The Einstein, Podolski, Rosen [EPR] article in 1935, "Can Quantum Mechanical Description of Physical Reality be Considered Complete?", shows by its title that incomplete description was on of Einstein's principal concerns. We suggest that Einstein's intuition that nondeterminism was due to incomplete description was right but that his hypothesis that passive hidden variables were the cause of incompleteness was wrong. We propose a stronger form of incompleteness due to unobservable effects of active secondary observers whose modeling requires at least three-object interaction (an observed system, primary observer, and secondary observer). The idea of incompleteness is similar to that of Einstein, but the mechanism of incompleteness is active external stimuli rather than passive hidden variables. We suggest that incompleteness modeled by MIMs in the domain of computation provides a model for incompleteness of quantum theory descriptions in the domain of physics.

Hidden variables are modeled by SIMs, while hidden interfaces are more expressive than hidden variables for the same reason that MIMs are more expressive than SIMs. Whereas nondeterminism of hidden variable models is due entirely to the observer's partial knowledge, nondeterminism of MIMs is due to uncontrollable external interactions. We conjecture that an experiment to test the hypothesis of objective determinism can be designed by extending Aspect's experiment based on Bell's inequalities to behavior tests for three entangled states, representing an observed system, primary observer, and secondary observer. Two entangled states are not sufficient to test for quantum nondeterminism. At least three entangled states are needed for nondeterminism to occur in the MIM model, and an experiment to test this hypothesis as an explanation of quantum nondeterminism requires three entangled states.

Classical physics has a SIM model of observation, where observers have complete control over the systems they observe, while quantum observers are better modeled a sequential observers of a MIM. Human observers as well as models of physical observers are inherently sequential. Multi-stream observers of MIMs that would allow MIM behavior to be viewed as deterministic are difficult to specify and implement.

Deterministic multi-stream MIM observers would require observation of boundary conditions distributed in both space and time for state transition systems or systems of differential equations. Classical physics assumes one-point boundary conditions observable by sequential observation, while quantum theory must extend the model of observation so that multipoint boundary conditions in space can be observed, and

interaction further extends requires boundary conditions distributed in time to be observable. Boundary conditions at a single point of time are said to be closed, while boundary conditions distributed in time are said to be open. Computational inadequacy in perceiving complete system behavior of MIMs parallels physical inadequacy in perceiving the complete distributed interactions of a physical object.

classical physics: sequentially observable systems, one-point closed boundary conditions
quantum physics: multi-agent (distributed) systems, multi-point open boundary conditions

The hidden-variable model is a sequential interaction model with the property that the observer and observed system together form a closed system during the process of observation. In the multi-agent model, the composition of the observed system with any single observer is an open system, with multiple interfaces to other observers. Openness of systems correspond to having multiple interfaces with incompletely specified multi-point boundary conditions. In physics every subsystem is automatically an open system because gravitational and electromagnetic forces at its boundaries act as asynchronous observers. Sequential models of observation adequate for classical physics break down for quantum physics because observers interact with open systems subject to unpredictable interactions (random noise).

The idea that God appears to be playing dice from the perspective of any individual observer because observers have limited powers of observation and control suggests a new meaning for the notion of omniscience. An omniscient observer is one who can observe global determinism for a locally nondeterministic world. Einstein would have approved of this explanation of observed nondeterminism by active secondary observers. Einstein's introduction of hidden variables corresponds computationally to the extension from algorithmic to sequential observation, while hidden interfaces correspond to the further extension to distributed observation, which is natural in the context of computation but radical in the context of physics.

Models of physics and empirical computer science have the common empirical goal of expressing the observable behavior of systems. This goal is expressed for sequential observers by mathematical models of non-well-founded set theory, coalgebra, and bisimulation. Classical physics, which assumes that observers together with the systems they observe can be treated as closed systems with no secondary observers, can be modeled by sequential observers. Quantum theory assumes that observed systems may be subject to perturbation by secondary observers, and is modeled by multi-agent interaction, which is less mathematically tractable than sequential interaction.

4.4 The Interactive Turing Test

Turing's two major contributions (Turing machines and the Turing test) appear at first to be unrelated, but can in retrospect be viewed as a part of a comprehensive unified model that relates inner computation and outer behavior. The Turing test model shows that Turing appreciated the importance of relating inner computation to external observable behavior.

However, limitations of Turing's model of computability are reflected in limitations of his model of behavior. Turing's published dialogs consider only unrelated sequences of questions, and not behavior for sequences of related questions or streams of questions (requests) from multiple questioners. Searle's Chinese room experiment likewise assumes translators act like TMs in following a fixed set of rules with no memory between successive translations. Eliza illustrates the great contribution that memory in the questioned system and follow-up by the questioner can make to the realism of a dialog, but requires both the questioner and system to be modeled by SIMs rather than TMs.

Turing's behaviorist model of thinking has been justifiably criticized as inadequate for two reasons:
intentional skepticism: TMs cannot adequately model inner (intentional) qualities of thinking
extensional skepticism: TM behavior is extensionally too weak to model thinking

However, the generalization that thinking cannot be modeled by behavior is an erroneous overreaction. The Turing test is deficient because TMs are too weak as a model of behavior, not because, as claimed by Searle [Se], behavior is an inadequate model of thought. The interactive Turing test not only augments the extensional behavior that machines can exhibit, but also their ability to express intentional qualities.

Searle's Chinese room argument assumes that rule-governed translators in the Chinese room perform their tasks noninteractively. Since his intent is to prove that machines cannot think he has no incentive to consider interactive adaptation during the machine translation process, though he would certainly have included adaptation in any model of human translation. His conclusion that algorithms have no intentionality do not apply to history-dependent translators that harness past experience or external experts (oracles) - something that interaction machines are certainly capable of doing. Searle was right about the inadequacy of the Turing test, but for the wrong reasons. By contrast, Turing had the right experimental design but applied it to the wrong model of computation.

Turing's question "Can machines think?" has different answers for different notions of "machine" and "think". The Turing test, not surprisingly, assumes question-answering machines to be Turing machines and equates thinking with algorithmic answering of questions by noninteractive computation. Turing's illustrative questions [Tu2] about arithmetic, chess, and composing poetry do not require the machine to remember earlier questions in answering later ones.

The interactive Turing test preserves Turing's behaviorist assumption that thinking is specifiable by behavior, but extends modes of questioning to be interactive. Analysis of interactive question answering yields behaviorist models of "thinking" with more expressive extensional and intentional behavior that provide realistic models of thinking and differ qualitatively from the traditional Turing test model.

algorithmic thinking: behavior of TMs that answers single questions by algorithmic computation
algorithmic Turing test: questioner asks single questions with algorithmically computable answers

sequential thinking: behavior of PTMs (SIMs) that answers sequences of history-dependent questions
sequential Turing test: questioner asks a sequence of follow-up questions, as in an interview

distributed thinking: behavior of MIMs that react to multiple autonomous streams of requests
multi-agent Turing test: global behavioral response to multiple autonomous requests

Algorithmic, sequential, and distributed thinking are progressively stronger forms of behavioral approximation to human thinking, defined by progressively more stringent empirical tests of behavior. Sequential thinking is realized by SIMs and observed by observers that ask follow-up questions (make sequential observations). Distributed thinking is realized by MIMs and requires multi-agent observers that can observe how the system responds to multiple potentially conflicting asynchronous requests.

Testing for multi-agent behavior extends the Turing test beyond SIMs to interaction with external resources like the Library of Congress, the Web, or human experts. Resources accessible to an IM are hidden from the questioner, who cannot distinguish whether the tested agent is an IM or a TM. The interactive Turing test extends machines without changing the form of the Turing test. Turing would certainly have accepted the sequential and multi-agent Turing tests as legitimate, and would have approved of the behaviorist notions of sequential and distributed thinking as conforming to the spirit of his notion of thinking.

Current IQ tests that focus on algorithmic intelligence by asking single questions have been shown to be poor predictors of later human success because they measure only limited forms of human problem solving. Sequential thinking can be tested by comparing the response of systems to sequences of related questions with human responses. Distributed thinking is tested by comparing system responses to autonomous streams with human responses. Sequential and distributed thinking require radical revision of the technology of intelligence testing. Devising "intelligence" tests for sequential and distributed thinking is a challenging task that raises interesting research issues and could correlate significantly better than current IQ tests with later human success. Studies have shown that tests that measure human ability to delay gratification, such as the marshmallow test [Go], are better predictors of later success than IQ tests.

The arguments of both intentional and extensional skeptics lose much of their force when the algorithmic Turing test is replaced by the sequential and multi-agent Turing tests. The weak extensional behavior of TMs causes them to be ineffective in modeling intensional behavior. SIMs and MIMs can express stron-

ger forms of intentionality because of their stronger extensional behavior. Searle's argument that machines cannot express intensional behavior is partly neutralized by SIMs that express intentionality through history dependence, and further weakened by MIMs that express the strong intentionality of multi-agent collaboration. The translators of Chinese stories in Searle's Chinese-room experiment must do more than follow transliteration rules in responding to interactive questions of sequential or multi-agent Turing tests. SIMs and MIMs have many, though not all, the properties that Searle calls intensional.

Penrose's extensional arguments that computers cannot express extensional behaviors [Pen] are valid for TMs and possibly for SIMs but not for MIMs [We3]. Penrose argues that the nondeterminism of physics may be illusory because physical phenomena may be deterministic but noncomputable. He suggest that nondeterminism arises because we limit our physical model to being computable and that a noncomputable model of physics could potentially resolve nondeterminism.

We agree with Penrose that TM models of computation are extensionally too weak to model physics. However, physics is computable according to the broader notion of MIM computability: Penrose's arguments about the noncomputability of physics by TMs become inapplicable when computing is broadened to include interaction. In fact, if Penrose's term "noncomputable" is replaced by the term "not-TM-computable", IMs can be viewed as the "not-TM-computable" models that Penrose needs, since they model the subjectively nondeterministic but objectively deterministic models that Penrose postulated.

In retrospect, Penrose's insight that nondeterminism of physics is due to a weakness of the model rather than an inherent feature of the physical world may turn out to be remarkably close to the truth. His mistake was to accept TM computability as the most powerful form of computing when in fact there are stronger models. By reformulating Penrose's argument in terms of non-TM computability and introducing MIMs as a stronger form of computation, we not only validate Penrose's argument but also demystify it by providing a concrete mechanism and model for the stronger form of computation. Whereas Penrose's hypothesis is not testable, we expect that the hypothesis that MIMs provide a model for quantum theory can be tested.

Penrose's view that computers cannot model physics is based on a misconception of the nature of computing shared by Church and Turing that has its historical roots in the rationalism of Plato and Descartes [We3]. By showing that interaction machines express empirical computer science, we show that arguments of Penrose, along with those of Church, Turing, and Descartes, are rooted in a common rationalist fallacy concerning the role of noninteractive algorithms in modeling the real world. Penrose's instinct that Turing machines cannot model physics is correct, but his belief that noncomputable models are the key to a unified theory of physical and mental phenomena rests on a limited definition of computability.

Though Searle's and Penrose's criticisms of algorithmic thinking become much weaker when applied to sequential and distributed thinking, the anti-behaviorist argument that inner processes cannot be unambiguously described by behavior remains valid, since behavior is an abstraction of system structure. Since behavior specifications have nonenumerable system implementations, behavior is a weak determinant of system structure. However, the status of behaviorist models is completely changed when thought is modeled by interactive behavior that is extensionally and intentionally richer than algorithmic behavior.

5. Specification of Interactive Systems

Section 5.1 defines specifications as constraints on observable behavior and shows that systems satisfying a specification are coinductively defined and generally nonenumerable. Section 5.2 places Godel's incompleteness result in perspective, showing that inductively defined logics are necessarily incomplete in expressing coinductively defined domains of objects. Section 5.3 relates modeling principles of empirical computer science to those of physics, showing that explanatory power of physical theories parallels expressive power of computational models. Empirical computer science provides an interdisciplinary framework for analysis applicable to physics and other empirical disciplines.

5.1 Specification and Possible Worlds

Specifications for both interactive systems and algorithms are constraints on observable behavior that specify observation equivalence classes. Algorithms are specified by sets of I/O pairs, SIMs are specified

by sets of sequences of I/O pairs, while MIMs support concurrent interaction that has no adequate formalism for specifying complete observable behavior.

Specifications Sp are defined by a set of observations O_{Sp} that determine necessary and sufficient behavior requirements for a system S to satisfy its specification. Sp determines a class C_{Sp} of systems S consistent with the specification. Sp completely specifies a system S if $O_{Sp} = O_S$ (Sp specifies all possible behavior of S) and partially specifies S if $O_{Sp} \subseteq O_S$. Partial specifications specify only required behavior and leave nonrequired behavior for systems S unspecified.

Consistency: A system S is consistent with a specification Sp if $O_{Sp} \subseteq O_S$, where O_S is the set of all possible observable behaviors of S. Sp determines a set $C_{Sp} = \{S \mid O_{Sp} \subseteq O_S\}$ of systems consistent with Sp.

C_{Sp} is the maximal class of systems satisfying Sp: specification classes C_{Sp} are coinductively defined. In this case maximality means that any S such that $O_{Sp} \subseteq O_S$ is consistent with Sp. Maximality imposes a constraint on behavior that expresses partial behavior and is consistent with a nonenumerable class of systems having the given partial behavior. Behavior specifications define services that systems can perform for users, but are incomplete, approximate system descriptions that do not try and cannot hope to fully describe either complete behavior or inner system structure.

The class C_{Sp} of systems satisfying a given specification Sp (for example, programs that realize a computable function) is generally nonenumerable. Software engineering is an art rather than a science partly because behavior equivalence classes C_{Sp} are nonenumerable and partly because noncompositionality precludes building up composite specifications from their components. The use of design patterns as primitves for system design is an ad hoc and nonformalizable process, providing evidence for the nonenumerability and noncompositionality of behavior specifications in building composite systems.

Software specification, verification, construction, and execution processes are realized by navigation in a conical space defined by an equivalence class and its representative specification elements (Figure 13).

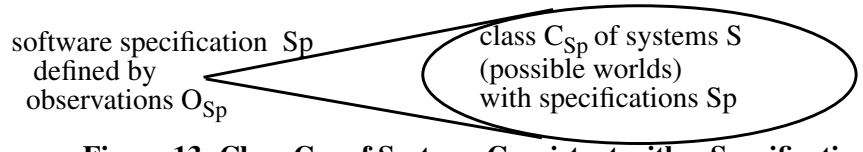


Figure 13: Class C_{Sp} of Systems Consistent with a Specification

Software construction and implementation starts from a specification Sp and maps it into an element S of C_{Sp}, while the converse process of specifying an already given system starts from a system S and aims to construct an associated Sp. Verification starts by being given both S and Sp and determines if S is consistent with Sp. The elements of the space and the direction of navigation differ for each of these processes.

This notion of specification is domain independent, applying equally to computing systems and physical systems that arise in nature. Specifications Sp determine an observation equivalence class C_{Sp} of consistent systems S that are said to be *possible worlds* for Sp. Software systems diifer from physical systems in that software designers model construction processes while physical systems are given in nature.

5.2 Incompleteness of Specification Languages

Specification languages determine a class of things that may be specified. First-order logic is a specification language for algorithms that is incomplete for specifying the behavior of interactive systems.

Completeness: A specification language SL is said to be complete for a class of systems C if it can completely specify the behavior of all systems $S \in C$. That is, if $\forall S \in C \; \exists \; Sp \in SL$ such that $O_{Sp} = O_S$. First-order logic is complete for functions effectively computable by TMs. That is, $\forall \; S \in TM \; \exists \; Sp \in fol$ such that $O_{Sp} = O_S$ (for all TMs its observable behavior O_{TM} can be completely specified in fol).

The inadequacy of first-order logic as a specification language for sequential interaction is a form of

Godel incompleteness. The inadequacy of non-well-founded set theory as a formalism for multi-agent systems could also be formulated as an incompleteness result. First-order logic (fol) specifies input-output behavior of algorithms by pre- and post-conditions, but is not strong enough to completely specify sequential or multi-agent interactive behavior. Godel's incompleteness theorem shows that fol cannot completely specify arithmetic over the integers. We adapt Godel's result to show that fol is an incomplete specification language for sequential interaction. The key argument is that fols can express only enumerable (inductively defined) behavior, while SIM behavior is expressed by nonenumerable non-well-founded sets.

Logics that derive theorems from axioms by rules of inference have inductively defined axioms and theorems that can prove only an enumerable number of facts (theorems) about the systems that they model. Reasoning that marches forward from given axioms to their provable consequences is a variant of the non-interactive TM paradigm of computation. It is too weak to reason about interactive behavior whose nonenumerable interactive computations are modeled by the nonenumerability of streams.

A formal system is *sound* if all its theorems are true and *complete* if all its true properties are theorems. Soundness and completeness restrict the true properties of models to have the same cardinality as the number of theorems [We3]. This limitation expresses an inherent weakness of traditional logic: any system with a non-enumerable number of modeled properties cannot be completely formalized.

Folk theorem (enumerability): Sound and complete formal systems can model only domains with an enumerable number of properties.

Proof: Let Tr be the set of true properties of a modeled domain and Pr be the theorems of a first-order logic for that domain (Figure 14). Soundness means that $Pr \subseteq Tr$, completeness means $Tr \subseteq Pr$, and soundness and completeness together mean Pr = Tr. Since Pr is enumerable, Tr must be enumerable.

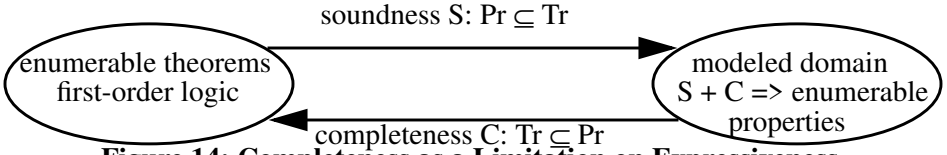


Figure 14: Completeness as a Limitation on Expressiveness

To prove incompleteness it is sufficient to show the domain we want to model has nonenumerable properties. Proving that a given modeled domain is nonenumerable is harder than proving the folk theorem, but diagonalization is a useful technique for proving that specific semantic domains are nonenumerable. Godel's incompleteness proof is essentially a diagonalization proof that the domain of arithmetic over the integers has a non-enumerable number of properties. Thus first-order logic is an inadequate specification language for completely specifying all properties of arithmetic over the integers.

Godel's incompleteness result and proof technique can be used to prove incompleteness not only for mathematical systems but also for interactive computing systems. The result that SIMs have nonenumerable computations is formally proved by showing that streams that model sequential computation are nonenumerable. The behavior of SIMs for all their possible computations cannot therefore be completely specified by traditional logic. Problems that have been independently proved to be non-enumerable, like program equivalence, automatically fall into the class of incompletely formalizable problems.

Incompleteness of interactive systems means that first-order logic is too weak to be a specification language for interactive behavior. First-order logic is a language of choice for weakest precondition specifications of algorithms. Behavior specifications for sequential interaction that require extension of precondition/postcondition specifications to assume/guarantee specifications for sequential interaction cannot be specified by first-order logic. They can, however, be specified by non-well-founded sets.

The existence of definition and reasoning formalisms for nonenumerable systems requires Godel's incompleteness result to be reinterpreted. Incompleteness becomes a result about the weakness of inductive (non-circular) reasoning rather than a fundamental weakness for all possible forms of formal reasoning. Reasoning by marching forward from known facts by truth-preserving rules is shown, by incompleteness, to be inadequate for reasoning about interactive systems. But non-well-founded set theory provides a formalism that extends both our definition and our reasoning ability to a larger class of domains,

including the domain of streams that model sequential interaction (Figure 15).

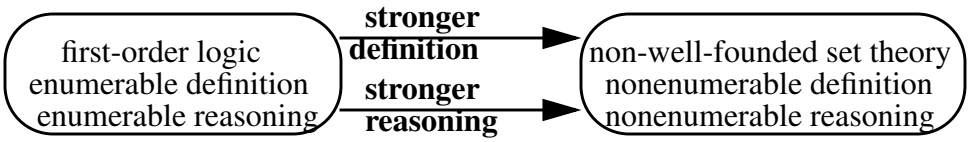


Figure 15: Extension of Definition and Reasoning Power

5.3 Empirical Models in Physics and Computation

The relation between behavior specifications and the systems they specify is central to both physics and computation. We show that physical theories are behavior specifications whose possible worlds correspond to systems consistent with a software specifications and whose explanatory power corresponds to expressive power of models of computation.

A physical theory T is a specification that determines an equivalence class of possible worlds consistent with predictions of T. T determines a set O_T of observable predictions. The predictions O_T in turn determine possible worlds PW_T consistent with T. Classes of possible worlds, like classes of systems consistent with specifications, are coinductively defined equivalence classes: $PW_T = \{W \mid W \text{ is consistent with } T\}$. The set of possible worlds for a consistent theory T is not enumerable.

Physical theory: A physical theory T specifies an equivalence class of possible worlds PW_T consistent with the theory: $PW_T = \{W \mid O_T \subseteq O_W\}$.

Explanatory power: A theory T is said to have greater explanatory power than T' if its predictions O_T explain (predict) a greater range of behavior than O_T'. If T has greater explanatory power than T', the class PW_T of possible worlds consistent with T is strictly smaller than those satisfying T'.

The parallels between software specifications of equivalence classes of systems and physical theories as specifications of equivalence classes of possible worlds is illustrated in Figure 16.

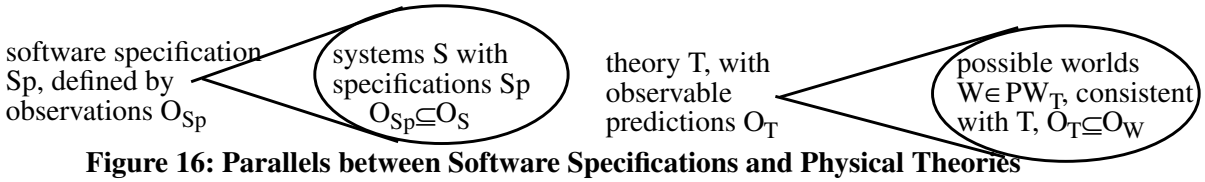


Figure 16: Parallels between Software Specifications and Physical Theories

Let R be a specification of all possible behaviors O_R of the real world. O_R includes observations by each of the five senses and by tools like telescopes, microscopes, and electromagnetic sensing devices. A theory T is said to be consistent with R if $O_T \subseteq O_R$. Though consistency of theories with R cannot be tested, the concept of O_R as a conceptual specification of observable real-world behavior is a useful one.

Since the ability of a theory to explain the real world is more important than its consistency we introduce the idea of relative explanatory power, which "projects" the set O_R of observations that define R onto the set of O_T of observations of the theory T, viewed as an observer of R in the sense of section 2.1.

Relative explanatory power: The *relative explanatory power* of a theory in explaining the real world R is $O_T \cap O_R$. A theory T has greater relative explanatory power for R than T' if $O_T \cap O_R \supseteq O_{T'} \cap O_R$.

When T and T' are consistent with R this reduces to $O_T \supseteq O_{T'}$. But the notion of relative explanatory power depends on features of R explained by a theory irrespective of whether it is consistent. Newton's theory with a large $O_T \cap O_R$ may be superior to a consistent theory with a smaller $O_T \cap O_R$. This example illustrates that inconsistent models can be useful in reasoning about the real world, and that using potentially inconsistent models is extends our reasoning ability in fundamental ways. This viewpoint takes Fin-

sler's Platonic philosophy of existence (section 3.2) one step further: according existential status to inconsistent theories extends our ability to reason about incompletely observable systems in the real world.

Established theories can evolve by a paradigm shift [Ku] into "better" theories with greater relative explanatory power for R. For example, observations concerning the bending of light rays by gravitational fields and of the perihelion of Mercury falsified Newton's theory and caused its replacement by a new theory (relativity theory) consistent with R for a greater range of behaviors. If T is relativity theory and T' is Newton's theory, then the observation falsifying Newton's theory is in $O_T \cap O_R$ but not in $O_{T'} \cap O_R$. Any observation in $O_T \cap O_R$ - $O_{T'} \cap O_R$ is a distinguishability certificate between T and T' for R.

Physicists would like to identify a unique real world W that causes the observable behavior O_A and is consistent with predictions O_T of a theory, but that is not possible through observations alone. The set of possible worlds W consistent with any theory is nonenumerable for the same reason that systems implementing a given behavior are nonenumerable. The choice of a unique W from among the set PW of consistent possible worlds predicted by a theory is realized by a non-empirical process called abduction.

Abduction: Abduction is the selection of a representative element of an equivalence class defined by a behavior specification (theory). It determes the inner structure of a black box from its behavior.

The process of abductive inference from predictions O_T of a theory to a choice of an actual real world $W \in PW$ is nonempirical (metaphysical), since it is not based on physical observation. Physicists have used Occam's razor as a selection criterion among behaviorally equivalent possible worlds [We3], for example in selecting the Copernican over the Ptolemaic model of the solar system. The use of abduction in inferring reality from observations and theories raises both epistemological and ontological questions:

epistemology: What is the *epistemological status* of abductively inferred knowledge? Are principles like Occam's razor or evolutionary principles like natural selection acceptable as the basis for abductive inference? What is the criterion that distinguishes acceptable from questionable abductive assumptions?

ontology: What is the *ontological status* of systems inferred by abduction? In particular, what is the ontological status of real worlds W inferred by abduction from observations and theories? Do *realists* who believe in the existence of a unique $W \in PW$ have a defensible position, or is realism merely a useful rather than a logically sound belief?

Ontological and epistemological questions have very different answers for software engineering than for physics. Software engineers answer the ontological question affirmatively, since they must bite the bullet and build a system that meets the specification. The epistemological question focuses on the effectiveness of design and implementation procedures rather than on the validity of abductive inference. Occam's razor is transformed from a principle about possible worlds to a principle about preferences for simple over complex designs. Software engineers construct actual realities under the control of the designer while physicists speculate about possible worlds consistent with observations. Descriptive principles about the nature of reality become prescriptive principles for design and implementation. Because software engineers build "real worlds" corresponding to specifications they can examine the relation between inner structure and behavior of systems in a way that physicists cannot, providing insights from computer science about the class of possible worlds consistent with physical predictions.

Software engineers have finer-granularity design criteria like modularity, maintainability, and portability for selection among functionally equivalent systems. Maybe the grand designer of the universe, or an impersonal natural selection law, makes use of software engineering qualities in selecting real-world mechanisms that cause behavior. However, the question whether scientific models should commit to unobservable implementation properties of reality is an ongoing matter of debate among quantum theorists.

Both physics and software engineering involve navigation in the conical design space defined by a specification (theory) and its equivalence class of implementations (worlds). The navigation paradigms differ in their initial conditions and goals. Software engineering develops systems starting from a given specification, while physics develops theories (specifications of regularities) starting from observations.

Physics (observation paradigm): Given an observable behavior O_R, construct a theory T whose

observable predictions O_T have an intersection $O_T \cap O_R$ with O_R that is as large as possible. There are open ontological questions about the existence of the real world R corresponding to the observable behavior O_R.

Software engineering (construction paradigm): Given a specification Sp, construct a system S whose behavior O_S is consistent with Sp ($O_{Sp} \subseteq O_s$). Select (construct) a system in the equivalence class O_{Sp}.

6. Conclusion

We are only beginning to understand how to formalize the richer behaviors expressible through interaction. Non-well-founded sets appear to be a stable extension of set theory for expressing sequential interaction. Further extension to handle distribution has not yet been mathematically formalized, but the idea that such interactions are definable by modifying the anti-foundation axiom suggests that distributed modes of interaction could in principle be set-theoretically formalized and proved consistent. This is one of many open problems to be addressed in developing a comprehensive mathematical theory of interaction.

The extension of models of computation from TMs to SIMs and MIMs is of more than theoretical interest, since both SIMs and MIMs capture important natural classes of problems. SIMs express two-agent interaction, including traditional object-oriented and agent-oriented sequential models, while MIMs express multiagent interaction and collaborative behavior. The existence of formal models for SIMs and of insights about how MIMs can be formalized suggests that the search for formal models of interaction is not as hopeless as it appeared even ten years ago.

Mathematical models of interaction bridge the gulf between formal models of traditional logic and empirical models of physics and the natural sciences. The anti-foundation axiom and yet to be defined variants for multi-agent interaction lay bare the mechanisms by which the gap between mathematical and empirical models can be bridged. The key step that underlies both greater modeling power and mathematical power is a shift from inductive to coinductive thinking.

Coinductive thinking involves a paradigm shift, not only in computer science but also in logic and mathematics, that extends the notion of what is formalizable from recursively enumerable sets specified by first-order logic to nonenumerable open sets that describe sequential and multi-agent interaction. It extends our ability to model real-world systems, bridging the gap between models of computation of Church and Turing and software engineers who build software systems as well as AI researchers who explore agent-oriented programming. Connections between computational expressiveness and physical explanatory power and between multiple interface models and quantum nondeterminism suggest that the significance of models of interaction extends beyond computer science to foundations of scientific modeling.

This paper has included the following contributions:

- defined sequential and multi-agent models of interaction in terms of stream transduction
- defined expressiveness of finite agents in terms of observation equivalence
- showed that greater expressiveness is due to stream domains rather than greater transformation power
- identified persistent Turing machines as a minimal extension of TMs for sequential interaction
- proved the existence of a hierarchy of levels of expressiveness for sequential interaction
- showed that multi-agent interaction machines are more expressive than sequential machines
- adapted the observation-based paradigm to model interactive specification
- characterized sequential models of interaction in terms of non-well-founded sets and coalgebras
- identified inductive definition as the common element of Turing equivalent systems
- identified greatest fixed point semantics as a key element of interactive models and contrasted it with least-fixed point semantics of algorithmic models
- showed that Godel's incompleteneness result expresses the limitations of inductive (constructive) mathematics in modeling stronger formalist and empiricist (interactive) formal systems
- developed extensions of the Turing test corresponding to extensions of Turing machines
- showed that expressiveness of interactive models is related to expalanatory power of physical models
- showed that nondeterminism in multi-agent models is related to quantum nondeterminism

7. References

[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.

[BS] Jon Barwise and Jerry Seligman, *Information Flow: The Logic of Distributed Systems*, Cambridge University Press, 1997.

[BV] Ethan Bernstein and Umesh Vazirani, Quantum Complexity Theory, *SIAM Journal on Computing*, October 1997.

[CT] Yi-Yen Chang and Roberto Tamassia, Dynamic Algorithms in Computational Geometry, *Proc IEEE*, Sept 1992.

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

[De] D. Deutsch, Quantum Theory, the Church-Turing Principle, and the Universal Quantum Computer, *Proc. Roy. Soc*., London Series A, 400, 1985.

[Di] Edsger Dijkstra, The Discipline of Programming,

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

[Fe] Richard Feynman, *Lectures on Computation*, Addison Wesley, 1996.

[FHMV] Ronald Fagin, Joseph Halpern, Yoram Moses, and Moshe Vardi, *Reasoning About Knowledge*, MIT Press, 1995.

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

[Ga] Robin Gandy, *The Confluence of Ideas in 1936, in The Universal Turing Machine - A Half-Century Survey,* Rolf Herken Ed., Springer Verlag, 1994.

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

[Go] Daniel Goleman, *Emotional Intelligence*, Bantam paperback, 1997.

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

[Kl] Stephen C. Kleene, Turing's Analysis of Computability, and Major Applications of it, in *The Universal Turing Machine - A Half-Century Survey,* Rolf Herken Ed., Springer Verlag, 1994.

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

[Ku] Thomas Kuhn, *The Structure of Scientific Revolutions*,

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

[MP] Zohar Manna and Amir Pnueli, *The Temporal Logic of Reactive and Concurrent Systems*, Springer Verlag, 1992.

[Pen] Roger Penrose, *The Emperor's New Mind*, Oxford, 1989.

[Per] Asher Peres, *Quantum Theory: Concepts and Methods*, Kluwer, 1993.

[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.

[Ru1] Jan Rutten, A Tutorial on Coalgebras and Coinduction, EATCS Bulletin 62, 1997.

[Ru2] Jan Rutten, Automata and Coinduction - An Exercise in Coalgebra, CWI SEN-R9803, 1998

[SB]Richard Sutton and Andrew Barto, *Reinforcement Learning*, MIT Press, 1998.

[Sch] Peter Schor, Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer, *SIAM Journal of Computing*, October 1997.

[Se] John Searle, *Minds, Brains, and Programs*, The Behavioral and Brain Sciences, 1980.

[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, Systems of Logic Based on Ordinals, *Proc. London Math. Soc.,* 1939.
[Tu3] Alan Turing, Computing Machinery and Intelligence, *Mind*, 1950.
[VW] Rob Van Glabeek and Peter Weijland, Branching Time and Abstraction in Bisimulation Semantics, *JACM*, May 1995, 43, 3, 555-600.
[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 Computation, Spring 1999, available at www.cs.brown.edu/people/pw.
[We4] Peter Wegner, Tradeoffs Between Reasoning and Modeling, in *Research Directions in Concurrent Object-Oriented Programming*, Eds. Agha, Wegner, Yonezawa, MIT Press 1993.