ELSEVIER

Theoretical Computer Science 192 (1998) 315–351

Theoretical Computer Science

# Interactive foundations of computing

Peter Wegner*

*Department of Computer Science, Brown University, P.O. Box 1910, Providence, RI 02912-1910, USA*

**Abstract**

The claim that interactive systems have richer behavior than algorithms is surprisingly easy to prove. Turing machines cannot model interaction machines (which extend Turing machines with interactive input/output) because interaction is not expressible by a finite initial input string. Interaction machines extend the Chomsky hierarchy, are modeled by interaction grammars, and precisely capture fuzzy concepts like open systems and empirical computer science. Computable functions cannot model real-world behavior because functions are too strong an abstraction, sacrificing the ability to model time and other real-world properties to realize formal tractability.

Part I of this paper examines extensions to interactive models for algorithms, machines, grammars, and semantics, while Part II considers the expressiveness of different forms of interaction. Interactive identity machines are already more powerful than Turing machines, while noninteractive parallelism and distribution are algorithmic. The extension of Turing to interaction machines parallels that of the lambda to the pi calculus. Asynchronous and nonserializable interaction are shown to be more expressive than sequential interaction (multiple streams are more expressive than a single stream).

In Part III, it is shown that interaction machines cannot be described by sound and complete first-order logics (a form of Godel incompleteness), and that incompleteness is inherently necessary to realize greater expressiveness. In the final section the robustness of interactive models in expressing open systems, programming in the large, graphical user interfaces, and agent-oriented artificial intelligence is compared to the robustness of Turing machines. Less technical discussion of these ideas may be found in [25–27]. Applications of interactive models to coordination, objects and components, patterns and frameworks, software engineering, and AI are examined elsewhere [28, 29].

The propositions P1–P36 embody the principal claims, while observations O1 through O40 provide additional insights.

*Keywords*: Turing machines; Interaction; Coordination; Time; On-line algorithms; Grammars; Process models; Games; Logic; Models; Incompleteness; Constraints; Emergent behavior; Empirical computer science

* E-mail: pw@cs.brown.edu.


*PII* S0304-3975(97)00154-0

# PART I: MODELS OF INTERACTION

Part I extends models of machines, grammars, and semantics from algorithms to interaction. We introduce interaction machines as an extension of Turing machines, extend phrase structure to interaction grammars, and show that interaction cannot be specified by state-transition semantics.

## 1. From Turing to interaction machines: Real numbers, real time, and real worlds

*Turing machines* (TMs) transform finite input strings into output strings by executing sequences of state-transition instructions (see Fig. 1). They are provided with a finite initial input string, but cannot accept external input while they compute.

**P1 (Turing machines):** TMs cannot model interaction since they shut out the world while computing.

Interactive systems are modeled by interaction machines (IMs) that are simple extensions of TMs.

**D1 (interaction machines):** IMs extend TMs by adding dynamic input/output (read/write) actions that interact directly with an external environment.

Interaction machines may have single or multiple input streams and synchronous or asynchronous communication, and can differ along many other dimensions, but all IMs are open systems that express dynamic external behavior beyond that computable by algorithms.

**D2 (interaction histories):** Observable behavior of IMs is specified by *interaction histories.*

**D3 (streams):** Sequential histories, called *streams*, are an interactive time-sensitive analog of strings.

Turing machine behavior is defined by computable functions on finite input strings, while interaction machine behavior is defined in terms of interaction histories (see Fig. 2). It is surprisingly easy to show that interaction machines cannot be reduced to Turing machines or realized by computable functions.

**P2 (interaction machines):** Interaction machines cannot be modeled by Turing machines.

Proposition 2 follows from the fact that finite input strings can always be interactively extended. IMs cannot be modeled by TMs with a finite initial input. They require an

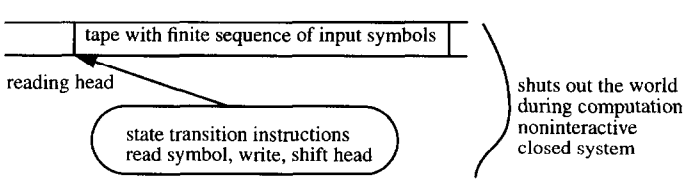


Fig. 1. Turing machine as a model for algorithmic computation.

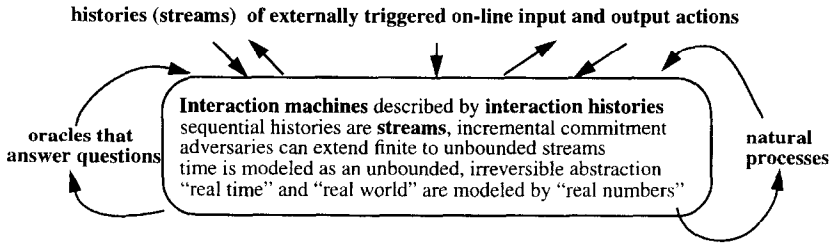


Fig. 2. Interaction machines and interaction histories.

infinite input tape whose input is not under the control of the IM and whose acceptance (termination) condition cannot be specified by a final state or by infinite repetition of a state, as in Buchi automata [22]. To establish P2 it is sufficient to show that IMs express mappings between infinite as well as finite strings.

Interactive behavior cannot be entirely captured by extending finite or infinite mappings of strings. Streams and histories are dynamically evolving partial structures that yield new finite forms of behavior not expressible by mappings of strings. The transformation semantics of strings is entirely defined by their elements, while that of streams (histories) may depend on time, adversaries, oracles, and protocols of interaction.

**O1 (streams):** The semantics of streams (histories) cannot be expressed by mappings (functions) from strings to strings.

Interaction cannot be expressed by or reduced to transformations (functions). Time is a nonfunctional property since the effect of functions (algorithms) does not depend on their computation time or on the time at which the effect occurs. Interaction extends computing to *computable nonfunctions* over histories rather than *noncomputable functions* over strings. Airline reservation systems and other reactive systems provide interactive services over time that cannot be specified by functions.

**O2 (behavior):** Interactive behavior includes nonfunctional finite behavior. Functions are too strong an abstraction that sacrifices the ability to model time and other real-world properties in the interests of formal tractability. They abstract away the ability to model autonomous external events, throwing out the baby with the bathwater.

Driving is a deceptively simple inherently interactive task that can, in the absence of traffic, be described by an algorithm specifying when a blindfolded person should press the accelerator, turn the steering wheel, or apply the brake. Driving home from work without traffic can, in principle, be modeled by off-line rules for closed systems that specify when to press the accelerator or turn the wheel. Driving home in traffic cannot be reduced to an algorithm even in principle: it depends on incredibly complex unpredictable on-line events that are not algorithmically or sequentially describable even for finite computations.

**O3 (persistence):** Persistent agent behavior over time is not algorithmically describable, since functions cannot express time and persistence as an inherently time-dependent property.

The correspondence between histories observable by a driver and the behavior they cause is not algorithmic even for discrete approximations of finite journeys because the set of all possible event histories is not algorithmically describable and the mapping of histories onto behavior is not algorithmic (like ants on beaches in Section 8). The idea that behavior of agents is determined by uncontrollable external interaction histories rather than by inner state transitions is central to models of interactive computing.

Smart bombs which interactively check observations of the terrain against a stored map illustrate the power of interaction. *Dumb algorithms* become *smart agents* (*embedded systems*) when enhanced by interaction. Algorithms are "dumb" and "blind" because they cannot interact while they compute: they are autistic in precluding interaction. In contrast, interactive systems model an external reality more demanding and expressive than inner algorithmic transformation rules.

**O4 (smartness):** Extending algorithms with interaction transforms dumb algorithms into smart agents.

The radical view that Turing machines are not the most powerful computing mechanisms has a distinguished pedigree. It was accepted by Turing, who showed in a 1939 paper [23] that Turing machines with oracles were more powerful than Turing machines. Milner [14] noticed as early as 1975 that concurrent processes cannot be expressed by sequential algorithms, while Manna and Pnueli [13] showed that nonterminating reactive processes like operating systems cannot be modeled by algorithms.

The intuition that computing corresponds to formal computability by Turing machines (a.k.a. the Church–Turing thesis) breaks down when the notion of what is computable is broadened to include interaction. Though Church's thesis is valid in the narrow sense that Turing machines express the behavior of algorithms, the broader assertion that algorithms precisely capture what can be computed is invalid.

**Thesis (intuitive computing):** Algorithms (Turing machines) do not capture the intuitive notion of computing, since they cannot express interactive computing and intuitive computing includes interaction.

Interaction machine inputs cannot be modeled by sets of finite strings both because streams cannot be modeled by strings and because finite sequences can always be extended. Interaction machines cannot be modeled by the set of all finite sequences (which is enumerable) but are more naturally modeled by infinite sequences, because adversaries have the last word and can always extend any finite sequence. The behavior of adversaries is better modeled by infinite processes that express the cardinality of the real numbers (Cantor diagonalization) than by enumerable sequences. The natural numbers are not closed with respect to interactive processes like diagonalization, just as the rationals are not closed under algebraic operations.

**O5 (closure):** Natural numbers are not closed under diagonalization, just like rationals under algebra.

**P3 (nonenumerability):** The interaction histories of an interaction machine are nonenumerable.

Real numbers were viewed in the 19th century as models of the infinite divisibility of continuous mathematical and physical space. Infinite divisibility of finite segments

of continuous physical space and infinite extensibility of discrete physical time give rise to dual nonenumerable abstractions of reality, corresponding to real numbers and real time. The set of all infinite digit streams is in one-to-one correspondence with both the real numbers and the input streams of an interaction machine. Interactive models bring out the serendipitous and unexpected connection between real numbers and the real world.

**O6 (real numbers):** Interaction machines model real time and the real world by the real numbers: models with infinite divisibility of space have the same cardinality as models with infinite extensibility of time.

## 2. Beyond on-line algorithms: Complexity, coordination, and constraints

Algorithms receive their input before the computation starts and produce a unique output after a finite number of noninteractive steps [11]. On-line interactive processes are not algorithms because they interact while they compute. But if an on-line process together with its environment forms a noninteractive (closed) system, then its interactive behavior can be specified by an on-line algorithm.

**D4 (closed-system property):** An on-line process P has the closed-system property if its accessible environment E can be described algorithmically and P with E forms a noninteractive (closed) system.

**P4 (on-line algorithms):** On-line processes with the closed-system property are on-line algorithms.

An on-line process P with the closed-system property may be viewed as the state transition mechanism of a Turing machine whose environment E is a finite tape. It performs algorithmic TM computations.

On-line algorithms for exploring graphs or maps have been extensively studied [18]. Their complexity is measured by interaction cost rather than instruction execution cost. The lower bound on interaction cost of finding a point on a line is the basis for complexity results of a family of related problems, such as finding a line in a plane or an intersection in a Manhattan graph.

**D5 (complexity):** Interactive complexity = number of interactive steps (see [18] for definition).

Agents for exploring static graphs have the closed-system property and are algorithms. This closed-system condition can be relaxed, so that the interactive behavior of agents that interact with systems evolving according to predictable algorithmic interactions can be described by on-line algorithms. But the behavior of agents that interact with open environments that change unpredictably during the process of computation inherently cannot be described by algorithms (see Section 6).

Two-person games model a form of interactive computing that has been extensively studied for both process models [15] and on-line algorithms. The result that "games against nature" with a polynomial number of moves have Pspace algorithmic

complexity [17] illustrates that interactive complexity, defined by the number of interactive steps, can be dramatically less than algorithmic complexity. Games against nature need an oracle to realize polynomial interactive complexity. However, for problems (like driving) in which local interactive responses yield acceptable global strategies, exponential off-line algorithmic complexity reduces to polynomial (linear) on-line interactive complexity without the need for oracles.

**P5 (complexity):** Under suitable locality conditions, problems with algorithmic complexity NP have interactive complexity P.

On-line algorithms are a restricted form of algorithmic computing slower than off-line algorithms because discovering known information requires computation time. The ratio between worst-case on-line and perfect-information off-line computation cost is called the *competitive ratio.*

However, interactive access to new information is more expressive than both on-line and off-line algorithms in providing new data to which the agent must react. Interactive systems that do not have the closed-system property respond to real-time events and express more than on-line algorithms.

**P6 (dynamic interaction):** Dynamic interaction is more expressive than on-line algorithms.

Interaction is a form of lazy (just-in-time) commitment to input values. Whereas enforced laziness restricts computation by withholding known data, just-in-time lazy access to dynamic data increases flexibility and expressiveness beyond that of algorithms. Dynamic interaction can be viewed as late (lazy) binding of inputs, in contrast to eager binding for noninteractive computation. Lazy binding of resources is a practical technique that explains the greater flexibility of interpreters over compilers. Inheritance owes its power to lazy binding of subclasses to parent classes. Mobile processes support lazy binding to resources as they become dynamically available.

**O7 (lazy binding):** Lazy interactive binding facilitates flexibility in interpreters, inheritance, and mobile processes.

Coordination [1] is the study of models, languages, and architectures that provide the glue for managing interactive behavior [4]. Glue is modeled as an active substance that captures properties of protocols, channels, pipelines, blackboards, and other architectural primitives of coordination media [28].

**O8 (coordination):** *Coordination* is the interactive analog of *transformation* for algorithms.

Transformation behavior is specified by functions built up from primitive instructions, while coordination behavior is specified by interfaces and protocols that can be nonalgorithmic and noncompositional. Coordination behavior cannot be specified by composition of primitives, but it can be specified by constraints on a space of "all possible behaviors". Constraints make no assumption about the behavior being constrained, specifying desired behavior by progressively constraining the superset of all possible behavior to a desired form, just as a sculptor progressively removes material from a block of marble until the desired form emerges [29]. Constraint specification is dual to constructive specification from primitives, capturing

emergent noncompositional behavior not expressible by constructive compositional techniques.

**O9 (constraints):** Constraints on histories specify interaction independently of state transitions.

**P7 (constraints):** Constraints can specify nonalgorithmic noncompositional emergent behavior.

The duality between inner bottom-up specification by logic and external top-down specification by constraints reflects the duality between algorithmic and interactive behavior description.

## 3. Extending the Chomsky hierarchy: From speakers to listeners

Automata are "listening machines" that recognize input sequences accepted by a state-transition mechanism, while grammars are "speaking machines" that generate strings specified by a phrase-structure grammar. The Chomsky hierarchy determines a correspondence between classes of listening mechanisms that recognize strings (automata) and classes of speaking mechanisms that generate them (grammars):

*finite automata* $\longleftrightarrow$ *regular grammars*

*pushdown automata* $\longleftrightarrow$ *context-free grammars*

*linear bounded automata* $\longleftrightarrow$ *context-sensitive grammars*

*Turing machines* $\longleftrightarrow$ *unrestricted grammars* (*recursively enumerable sets*)

Interaction machines extend the listening power of automata, but there is no generative grammar that correspondingly extends speaking power. The listening paradigm is more naturally extensible to interaction than the speaking paradigm, since input (listening) is a cause of interactive expressiveness while output (speaking) is a consequence of input. Chomsky's correspondence between automata and grammars abstracts away differences between listening and speaking that may legitimately be ignored for TMs but are necessary for IMs. Interactive extensions of the Chomsky hierarchy require distinctions between models of listening and speaking to be reintroduced.

**P8 (grammars):** Interactive listening machines can express richer behavior than generative grammars.

Interaction grammars (see Section 4) are a radical extension of generative grammars, while interaction machines are a simple and natural extension of Turing machines. Machines are a more powerful paradigm of computation than grammars, since they can be extended more naturally from algorithms to interaction.

Automata have a state transition mechanism and a tape whose rules of engagement are noninteractive:

*automaton* = *state transition mechanism* + *noninteractive tape* (*finite initial string*)

Automaton classes of the Chomsky hierarchy differ in the nature of the rules of engagement for accessing their tape. Finite automata require the tape to be read-only, pushdown automata permit an unbounded auxiliary pushdown tape, linear-bounded

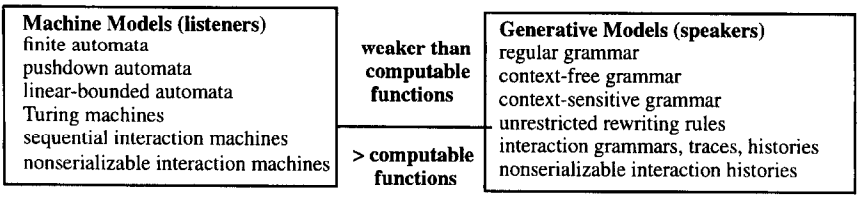


Fig. 3. The extended chomsky hierarchy: machines and associated behavior.

automata permit writing but place length bounds on the tape, while Turing machines permit unrestricted but noninteractive tape access.

Interaction machines explore the consequences of relaxing the restriction that automata are controlled by noninteractive tapes. We show that interaction enhances expressiveness, but that different forms of interaction have observably different expressiveness. In particular, sequential interaction controlled by a single stream is less expressive than nonserializable interaction of multiple streams.

**O10 (multiple streams):** Multiple streams are more expressive than a single stream.

Whereas the expressiveness of Turing machines is not increased by adding multiple tapes, the expressiveness of interaction machines may be increased by adding multiple streams. Different forms of expressiveness associated with asynchronous and nonserializable interaction are discussed in Section 11.

## 4. Interaction grammars: Incremental commitment, bisimulation and game semantics

**D6 (Generative grammars):** G = (N, T, S, P) specifies the syntax of languages L(G) over an alphabet of terminals T in terms of nonterminals N, a starting nonterminal symbol S, and productions P.

**Example:** *grammar for binary strings*: *string* → *empty*|0.*string*|1.*string* -- *generates binary strings*

"." *is string concatenation, while* "|" *is set union*

Interaction grammars replace generative production rules P for strings by reduction rules R for accepting histories (special case streams). Reduction rules cannot express interaction histories for problems like driving, but can express restricted forms of interaction like that associated with process models or two-person games. Interaction grammars for this restricted form of interaction have terminals (input symbols) and nonterminals (listening states including the initial state S).

**D7 (Interaction grammars):** IG = (N, T, S, R) accepts histories (streams) by reduction rules R with a dynamic "listening" operator "." and a "nondeterministic choice" operator "+". Machines specifiable by grammars IG will be called IG machines. The stream grammar below generates infinite streams, expressing servers (reactive systems) that react to a continuing stream of 0's and 1's over time.

**Example:** *grammar for streams*: *stream* $\rightarrow (0+1).stream$ -- *accepts binary streams* "." *listens for externally controlled input, while* "+" *expresses incremental choice and commitment*

Streams for this class of interaction grammars have the form "init.future(init)" where init is a past history, "." represents the present, and future(init) is a future set of possible worlds. Sequential histories have a past that is a string and a future that is an unbounded tree. The tree-structured future is incrementally transformed into a sequential past at each computational step. An input action moves one step down the tree, lengthening the init string by the current input and pruning the future tree by discarding paths not taken. The interactive "." operator "listens" for an input while "+" irreversibly commits to an actual event chosen from possible worlds when inputs arrive.

**O11 (time):** Interaction grammars IG capture the unboundedness and irreversibility of time:

*unboundedness*: *because the next step is externally controlled*

*commitment and irreversibility*: *by incremental actualization of possible worlds*

Languages L(IG) for interaction grammars IG specify sets of streams (possible worlds) whose semantics differs from strings in capturing the unboundedness and irreversibility of time. Expressiveness for grammars IG has both a static set-inclusion component and a dynamic component captured by the preservation of freedom of choice. An IG machine M1 is more expressive than M2 if for every past history init accepted by M2, the set of future histories future(init) of M1 includes the future histories of M2. Such incremental inclusion of future histories for all past histories is called dynamic inclusion.

**D8 (dynamic inclusion):** A collection of histories H1 *dynamically includes* histories H2 if H1 has at least as much freedom of choice as H2 for every initial history of H2.

Dynamic inclusion refines set inclusion to include incremental preservation of freedom of choice. The reductions a.(b+c) and a.b + a.c specify the same set of two acceptable strings, but the first dynamically includes the second since it preserves the freedom to choose between b and c after receiving a. Dynamic inclusion of histories is a natural basis for a definition of interactive expressiveness.

**D9 (expressiveness for sequential (serializable) interaction machines):**

M1 *is more expressive than* M2 *if and only if its histories dynamically include those of* M2

M1 *is observably equivalent to* M2 *if and only if each dynamically includes the other*

Dynamic inclusion is a sufficient as well as a necessary condition for IG-machine expressiveness. It specifies the fuzzy notion of observability by a precise notion of incremental freedom of choice, which is in turn defined by properties of the operators "+" and ".". It provides a working definition of IG-machine observability, fixing the granularity of abstraction for observational distinctions.

**P9 (inclusion):** Dynamic inclusion refines set inclusion as a measure of expressive power.

Preserving freedom of choice implies lazy commitment (never commit today to something that can be decided tomorrow). Lazy commitment is a useful military strategy that exploits premature commitment by the enemy. Financial options to exercise future choices have a market value. In two-person games the ability to delay commitment is the key to a winning strategy.

**O12 (laziness):** Lazy input → lazy commitment → dynamic inclusion → two-person games.

Interactive expressiveness can be modeled by a two-person game in which player 1 controls machine M1, player 2 controls machine M2, and player 2 wins if she can find a sequence of moves on M2 that cannot be replicated by player 1 on M1. Player 2 may be viewed as an omniscient adversary who looks for a move that cannot be matched by player 1 at each step. If player 2 cannot find a sequence of moves to defeat player 1, then player 1 wins and M1 is as expressive as M2. This game expresses dynamic (lazy) commitment because player 2 defeats player 1 if she has greater freedom of choice at any move. If player 1 can replicate any sequence of moves of player 2, then the machine M1 is said to simulate M2.

**D10 (simulation):** M1 simulates M2 if M1 can dynamically match every step of M2.

**D11 (bisimulation):** Bisimulation is the condition that M1 simulates M2 and M2 simulates M1.

The idea of observational equivalence as mutual simulation (bisimulation) is due to Milner and was formalized by Park [15]. Two machines that simulate each other are observationally equivalent in the sense that they can make the same observational distinctions. Bisimulation has finer-granularity observational discriminating power than set inclusion, capturing dynamic distinctions among streams specified by an interaction grammar that are indistinguishable as strings. It provides a temporal dimension for distinguishing among histories that captures the semantic notion of preserving freedom of choice.

**P10 (expressiveness):** Bisimulation, dynamic inclusion, and game semantics are equally expressive.

The power of game semantics to capture bisimulation and dynamic inclusion is surprising but natural because two-person games express the tension between inner and interactive cleverness of players and opponents who represent all possible behavior, including worst-case behavior of adversaries. Though it refines set inclusion to express time-sensitive semantics, the semantics of games and IG machines must be further extended to capture the more intricate multiple-input-stream behavior of driving or airline reservation systems.

## 5. State-transition versus observation structures: Intensional versus extensional behavior

Interactive behavior cannot in general be expressed by state-transition behavior because differences of scale and granularity may be too great. Human cognitive behavior

cannot be expressed by laws of chemistry or by the motion of electrons. The mapping between inner and externally observable behavior of agents may be arbitrarily complex, depending on the granularity of abstraction. The irreducibility of "observation structures" to "state-transition structures" mirrors irreducibility interaction to algorithms. *Extensional* observable effects cannot be captured by *intensional* operational semantics of state transitions:

**D12 (extensional and intensional behavior):**

*The intensional behavior of an agent is modeled by a state-transition system*

*The extensional behavior of an agent is expressed by interaction histories or streams*

**P11 (irreducibility):** Extensional behavior cannot express intensional behavior and vice versa.

*observable behavior of components is not expressible by inner behavior specifications*

*interaction semantics is not expressible by state-transition semantics or vice versa*

The difference in viewpoint between inner and observable behavior corresponds to that between operational and denotational semantics. Algorithms have an operational semantics specified by a formal correctness criterion, while correctness of interactive systems is unspecifiable even in principle:

*algorithms are formally specifiable though correctness of operational semantics is undecidable*

*interactive denotations cannot be specified by first-order logic or by state-transition semantics*

Automata are carefully defined to preserve a one-to-one correspondence between state-transition and input–output actions. This assumption of a one-to-one correspondence between observable input–output actions and inner state-transition actions must be abandoned for large-granularity interactive abstractions. State-transition descriptions (operational semantics) are inadequate for describing the observable interface behavior of objects and interaction machines. Interface operations of interactive systems are implemented by inner action sequences that may take a significant time or be nonterminating.

The observation structures of [5] provide a formal model of observability that distinguishes between state-transition and observation structures. But [5] uses state-transition structures as a scaffolding decorated by observation structures as a basis for defining observation semantics. This limits expressive power because it requires observations to be viewed through constraints imposed by state transitions. Observation and bisimulation structures defined by direct modeling of what is observed determine an inherently richer class of structures and semantic theories than state-transition structures. Structured operational semantics (SOS) specifies behavior at the wrong level of abstraction, since intensional structures are inadequate as a scaffolding for expressing extensional patterns of observable behavior.

The distinction between state-transition and interface-event descriptions of behavior is particularly significant for concurrency. Interleaving models that reduce

concurrent to nondeterministic sequential computation may be defined by the condition "tr1|tr2 = tr1.tr2 + tr2.tr1" for all transitions tr1, tr2. The corresponding condition "a|b = a.b + b.a" among interface events does not generally hold because state transitions may interfere. Concurrent execution "P|Q" of two processes does not in general have an interleaving interpretation as "P.Q + Q.P" because "." is not a meaningful operator for interactive components.

**O13 (noninterleaving):** Noninterleaving models of concurrency fall into two classes:
*enhanced operational semantics*: *constraints on inner behavior exclude some interleaved behaviors*
*true concurrency*: *inherently concurrent, noninterleavable behavior*

Causality, locality, and other constraints on interleaving behavior are examined in [6] in the framework of enhanced operational semantics, which, as its name implies, considers concurrency at the level of state transitions. True concurrency, which more radically violates interleaving semantics by admitting inherently concurrent behavior, is closely related to nonserializability of transactions (see Section 9).

**P12 (concurrency):** Interleaving models, enhanced operational semantic models, and true concurrency have progressively greater expressive power.

Interleaving models are expressible by sequences (traces) of interface events, enhanced operational semantics imposes constraints on trace sequences, while true concurrency expresses interface behavior not describable by traces. Greater expressiveness of nonserializable over serializable histories (discussed in Section 11) demonstrates the greater computation power of true concurrency over interleaving.

The semantics of process models is expressed extensionally by traces. Traces are used in debugging to expose the noninteractive steps of an algorithm to interactive scrutiny by users. The term "traces" describes both sequences of observations of a process and debugging traces, since both permit external interaction at each step. Debugging traces transform algorithms into interactive processes with greater expressive power. The intuition that interaction at debugging checkpoints increases expressive power is formally proved by showing that interaction machines are more expressive than Turing machines.

**O14 (debugging):** On-line debugging causes algorithms to behave like interactive processes.

Constructing inner models by outside-in inference methods like abduction [19] is at best a guess, since many inner mechanisms can cause the same observable behavior. Since programming is concerned with building systems inside-out, starting from inner state transition models, while specification is concerned with the outside-in description of what is to be built, computer science is centrally concerned with bridging the chasm between outside-in and inside-out description. However, the gap between inside-out and outside-in behavior is formally unbridgeable (neither can be uniformly expressed by or reduced to the other).

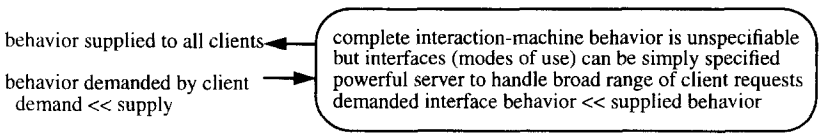


Fig. 4. Supplied server behavior versus demanded client behavior.

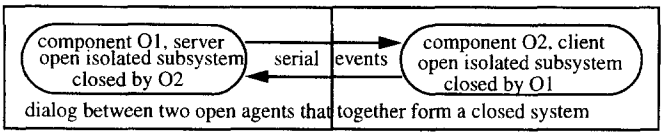


Fig. 5. Component composition as a constraint on behavior.

## 6. Open systems: Nonmonotonicity, duality

The fuzzy concept "open system" is precisely defined in terms of interaction.

**D13 (open, closed):** A system is open iff it is interactive and closed iff it is non-interactive.

Closed systems can become open by removing their parts. Interactiveness is *nonmonotonic*, since decomposition of closed noninteractive systems creates open subsystems and composition of open systems may produce closed systems. In contrast, concurrency and distribution are monotonic: all subsystems of nonconcurrent nondistributed systems also have this property. Nonmonotonicity implies that noninteractive (algorithmic) systems may have interactive subsystems whose behavior is non-algorithmic.

An engine of a car may behave unpredictably when a spark plug is removed. Animals (or persons) with an established behavior routine may behave erratically in unfamiliar environments. Subsystems with predictable (algorithmic) constrained behavior have unpredictable (nonalgorithmic) behavior when the constraint is removed and a greater range of possible behaviors must be considered:

**P13 (nonmonotonicity):** Openness and interactiveness are nonmonotonic system properties.

The fact that algorithms are not closed under the operation of taking subsystems is a lack of robustness, similar to the lack of closure of integers under division and of rationals under square root. The relation between closed systems and their open subsystems is analogous to that between rationals and reals.

Interaction machines provide a precise framework for the specification of openness and open systems. Openness of server components allows them to interact with clients over time. Open systems are designed to handle all clients, while individual clients often have simple interface demands. The supplied behavior of interactive systems is intractable, while demanded behavior is often quite tractable (see Fig. 4).

An interactive isolated component O1 becomes a noninteractive system when composed with a second component O2 so that each talks only to the other (see Fig. 5). Each component in isolation interacts freely with clients, while each component of the composite system constrains the behavior of the other:

The set of all possible behaviors of O1, viewed as an isolated system, is nonalgorithmic. When input to O1 is constrained to be that produced by O2 in response to a message stream from O1, O1's behavior can be tractably described. The component O2 constrains O1 to closed-system behavior. Algorithmicity is not preserved under decomposition of the system compose(O1, O2) into its components O1 and O2, and conversely nonalgorithmicity is not preserved under system composition.

The relation between O1 and O2 is asymmetrical for observers residing in one of the objects or when the two objects play different roles, as in client/server or agent/environment systems. Two-component systems where each acts as a constraint on the other arise in control theory, where isolated systems having richer uncontrolled than controlled behavior are natural. Composition that causes an open system to become closed is a special case of composition that yields open systems that can be further composed.

There is a duality between uncontrollability by O1 of its environment O2 and the uncontrollability by O2 of the inner actions of O1 [7, 28]. Inner actions of O1, called tau moves in [15], cannot be controlled by O2, just as outer behavior of O2 cannot be controlled by O1. Inability to control inner behavior of components leads to nondeterministic output actions by components on the environment, just as inability of components to control their environment is responsible for nondeterministic input actions.

**P14 (duality):** Observation/control duality in control theory mirrors algorithm/interaction duality.

## 7. Noncompositionality: Specification by constraints, emergent behavior

Interactive composition "P|Q" of two processes P and Q differs fundamentally from composition "P;Q" of two procedures. Whereas "P;Q" specifies sequential composition of transformations whose only interaction is passing the baton from P to Q as in a relay, "P|Q" specifies composition of persistent subsystems with multiple interactions over time. Milner defines composition "P|Q" as "*P and Q acting side-by-side, interacting in whatever way we have designed them to interact*", recognizing that observable interaction rather than unobservable concurrency is the property that process calculi must elucidate.

The term "interactive composition" better expresses component composition than "concurrent composition", since interaction can be observed while concurrency is unobservable and is in any case not a required property of interactive systems. Both processes and transactions are better expressed in terms of observable interactive properties than by unobservable state-transition steps of concurrent execution.

**O15 (composition):** Process composition is better modeled by noncompositional interaction than compositional state transitions.

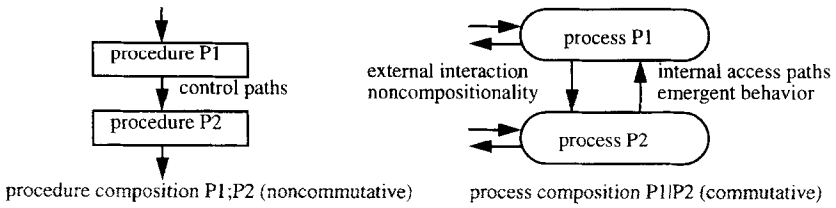


Fig. 6. Composition of procedures (algorithms) and processes (systems).

"P;Q" is noncommutative but compositional, while "P|Q" is commutative but noncompositional. It creates new (emergent) behavior whose whole is greater than the sum of its parts:

$behavior(P1;P2) = behavior(P1)$ *followed by* $behavior(P2)$

$behavior(O1 \mid O2) = behavior(O1) + behavior(O2) + interaction(O1, O2)$

Whereas gluing together (composing) algorithms yields an algorithm, gluing (coordinating) components to form a subsystem is not compositional: the whole is greater than the sum of its parts.

**P15 (noncompositionality):** Interactive behavior of processes and persistent components is not compositional.

The bad news that composite behavior is not expressible by component behavior is balanced by the good news that "emergent" behavior can enhance expressiveness. The irreducibility of composite wholes to their component parts is a feature of processes, objects, and systems (see Fig. 6), just as "emergent behavior" is a feature for neurons in the brain [16] and for quantum computers [3]. The fuzzy notion of emergent behavior may in fact be precisely defined as noncompositional behavior.

Compositionality is considered beneficial and indeed essential for constructive mathematics, but is harmful for expressiveness since it constrains the behavior of composite systems to be no more than that of their components. Dijkstra's well-known article "go to considered harmful" [8] took it for granted that formalizability was a primary goal and that interference with this goal was harmful. However, there is a trade-off between formalizability and expressiveness, so that features harmful to formalizability may be beneficial for expressiveness. In focusing on expressiveness rather than formalizability we invert the metric by which harmfulness is measured and view overemphasis of formalizability as harmful to expressiveness. Harmfulness depends on the goals and point of view of the client:

**O16 (behavior):** Compositionality is beneficial for formalization but harmful for expressiveness. Emergent behavior of processes, agents, and software components is inherently noncompositional.

Compositionality is a nice mathematical property that allows composite structures to be specified by "clean" reductionist techniques of mathematical logic and model theory. Declarative systems like the lambda calculus are carefully constructed to be compositional. But compositionality for structures defined by functions or algorithms breaks down for persistent components that interact over time.

Harnessing already-created components for useful purposes is dual to building a composite structure from primitives. Constraints express the sacrifice of freedom for discipline to realize collaborative component behavior for both software components and people who become cogs (algorithms) in corporations. Marriage constrains the freedom of individuals in order to create a collaborative family unit with goal-directed behavior. Application frameworks are realized by constraining behavioral freedom of classes in class libraries to realize collaborative behavior [29].

**D14 (frameworks):** A framework harnesses library components to realize goal-directed behavior.

**P16 (frameworks):** Frameworks can be specified by constraints on constituent components.

Viewing composition as a constraint on interactive behavior provides a systematic basis for modeling, dual to algorithm composition. Each interactive composition step constrains the behavior of composed components to a subset of their free behavior. Specification by constraints uses the counterintuitive contravariant principle "less is more" as a basis for analysis and design. Constraints are more powerful because they are applicable to noncompositional behaviors. Michelangelo's sculptures, realized by chipping away a marble slab, could not have been realized by gluing together small bits of marble.

## PART II : VARIETIES OF INTERACTION

In Part II we examine a variety of interactive models, ranging from interactive identity machines to process models and asynchronous and nonserializable interactive systems. We show that concurrency and distribution are orthogonal to interaction, provide new perspectives on process models, and explore a hierarchy of different levels of interactive expressiveness.

## 8. Interactive-identity machines: Pure interaction, judo, and reusability

**D15:** Interactive-identity machines (IIMs) immediately output their input without transforming it.

IIMs show that pure interaction without any computation can express very powerful behavior. IIMs are simple transducers that realize nonalgorithmic behavior by harnessing the computing power of the environment. They employ the judo principle of using the weight of adversaries (or cooperating agents) to achieve a desired effect. IIMs can be specified in a variety of language notations:

**loop** input(message); output(message); **end loop** -- specification by a loop

**while** true **do** echo input **end while** -- specification by a while statement

P = in(message).out(message).P -- recursive specification as a concurrent process

Mathematically, IIMs are the interactive analog of free algebras. Identity allows maximal scope for making observational distinctions among outputs, for the same reason

that free algebras have maximal homomorphisms onto subalgebras. Both interaction machines and free algebras preserve maximal distinctions among inputs in their outputs. Computation decreases observational differences through many-to-one mapping, just as homomorphism decreases algebraic distinctions. The free behavior of IIMs can be progressively constrained by computation to realize restricted forms of behavior that perform specific tasks, but observational distinguishability is greatest when no computation whatsoever is performed.

IIMs trivially model Turing machines by simply echoing their behavior.

**P17 (identity):** Interactive identity machines can express richer behavior than Turing machines.

IIMs can echo the behavior of both computable and noncomputable processes in the environment. Echoing or imitating behavior is a powerful technique of problem solving in both people and computers. The behavior of Eliza, called "echo intelligence", can be realized by IIMs with simple echo rules.

**O17 (Eliza):** Eliza uses interactive identity in simulating dialogs between patients and psychiatrists:

*Person*: *I am often very depressed.*

*Computer*: *Is it because you are often very depressed that you came to see me*?

It is easy to dismiss echo intelligence as an illusory trick having nothing to do with computing or real intelligence, but in the context of interactive models of computation we see that echo intelligence (reusability) is an important principle, framework, and mechanism for practical problem solving. Telephone receivers are in a very real sense as expressive as the conversations they transmit.

**O18 (reusability):** IIMs and echo intelligence realize problem solving by reusability.

The behavior of ants on beaches [21, 27], can be specified as the composition of an ant and a beach process by an interactive composition operator "||":

*ant* || *beach*

The beach constrains the ant to specific goal-directed behavior. The ant traces out a sequential path that is in principle completely determined once the ant is placed on a particular point of a particular beach. Though beaches are closed systems, beaches are not algorithmically describable and ant-beach systems do not have the closed-system property. Paths of ants on beaches are not algorithmically describable.

**P18 (agents):** Agents interacting with nonalgorithmic systems have nonalgorithmic behavior.

The problem of driving home from work has a similar structure to that of the ant finding its way home to an ant colony. It has the form "driver || city" where the driver plays the role of the ant and the city plays the role of the beach. Though the abstraction of a city adequate for purposes of driving is algorithmic, driver-city systems do not have the closed-system property because of dynamic interaction with other cars and are nonalgorithmic. The composite system "driver || city" is an open system since the city plan does not model traffic, while the system "ant || beach" is a closed system since beaches are presumed unchanging over time (though they are changed by tides and storms).

An interaction machine (or person) knowing no chess can win half the games in a simultaneous chess match against two masters by echoing the moves of one opponent on the board of the other [27]. Chess machines make use of intelligent input actions through one interface to deliver intelligent outputs through another. They harness the intelligence of one player to respond intelligently to a second player. Chess machines have the closed-system property and are in principle algorithmic, but because their algorithmic complexity is intractable, players are modeled as interactive systems that solve algorithmically intractable problems by heuristically acceptable interactive techniques.

The composite behavior of psychiatrists and patients has the form "doctor || patient". The composite behavior of an interactive chess machine with a player has the form "chess || player". In each case simple interactive processes have complex behavior because they are composed with an environment that is nonalgorithmic or open or (in the case of chess) algorithmically intractable. Interactive identity machines express the "pure" case of a transparent identity process I that takes on the behavior of a process with which it interacts. IIMs act as the identity element of process algebras:

$$I \,||\, process = process$$

Interactive identity machines realize the "management paradigm": managers harness and coordinate workers and resources in the environment without necessarily understanding them in precisely the way an interaction machine harnesses behavior in its environment. The composition of a manager $M$ with $m$ workers $Wi$ and $n$ resources $Ri$ can be represented algorithmically or interactively:

$$M(W1,\ldots,Wm,R1,\ldots,Rn) = M\|W1\|\ldots\|Wm\|R1\|\ldots\|Rn$$

The right-hand side expresses the traditional view of managers as algorithmic coordinators of workers and resources. The left-hand side, which views managers as processes that interact through worker and resource processes, expresses a richer class of potential behaviors.

**P19 (management):** Interactive management is more expressive than rule-based management.

The expressiveness of IIMs is dependent on external resources, while algorithms are self-reliant. High achievement, whether by machines or people, can be realized by self-sufficient inner cleverness only for "small" problems: scaling up to "large" problems requires harnessing the environment. Collaborative achievements of embedded computers, corporations, or nations are dependent on the effective use of an environmental infrastructure (it takes a village!). Inner restructuring techniques like structured programming are not as scalable as interactive restructuring. Interactive systems are more scalable than algorithms not only in computing but also in physics and social engineering.

**O19 (scalability):** Large-scale restructuring by CEOs and presidents of large organizations is interactive, focusing primarily on interaction among subunits of the organization.

## 9. Interaction, parallelism, distribution: Correctness conditions for transactions

Parallelism and distribution specify inner structure, while interaction specifies interface behavior.

*interactive systems interact with an external environment that they do not control*
*parallelism (concurrency) is the property that computations of a system overlap in time*
*distribution is the property that components of a system are geographically or logically separate*

**P20 (orthogonality):** Interaction, parallelism, and distribution are orthogonal forms of behavior.

Interaction may occur without parallelism or concurrency, as in IIMs. Parallel algorithms for matrix algebra, graph reachability, or maximum flow are noninteractive, receiving all their input before the computation starts. They increase execution speed and computational richness but not expressiveness.

Noninteractive distributed systems likewise increase computational richness but not expressiveness. Distributed algorithms for leader election or consensus as described in [12] are noninteractive, being concerned with processes that interact among themselves but not with an external environment. The many rich and worthwhile textbooks and courses on parallel and distributed algorithms focus without exception on noninteractive problems to ensure that the computations being analyzed are algorithmic.

**O20 (algorithms):** Parallel and distributed algorithms are noninteractive.

On-line processes are algorithms only when the process together with its environment is closed. Discovering the shortest path in a graph or a map for a finite fixed environment is an on-line algorithm because the agent together with its environment is closed. The complexity of on-line algorithms for agents that construct internal models of a fixed finite external world is analyzed in [18] and other papers. However, the problem becomes an open nonalgorithmic problem if edges of the graph can be deleted by a malevolent adversary (roads can be closed because of snow). Though the development of on-line algorithms for closed finite environments is a challenging research problem, interactive object-oriented and agent-oriented models of software engineering and AI are inherently nonalgorithmic.

The recognition that interaction rather than parallelism or distribution is the key element in providing greater behavioral richness is a nontrivial insight that requires a reappraisal of the role of parallelism and distribution in complex systems. The horizontal base plane of Fig. 7 includes many interesting and important algorithmic systems, while systems not in the base plane are nonalgorithmic.

Modeling transaction correctness by concurrency expresses this problem at the wrong level of abstraction, since serializability is an implementation-dependent criterion of correctness. Viewing transactions as atomic noninteractive units substitutes *interaction control* for *concurrency control* as a correctness criterion. Noninteractiveness is locally checkable, while nonserializability is a global condition.

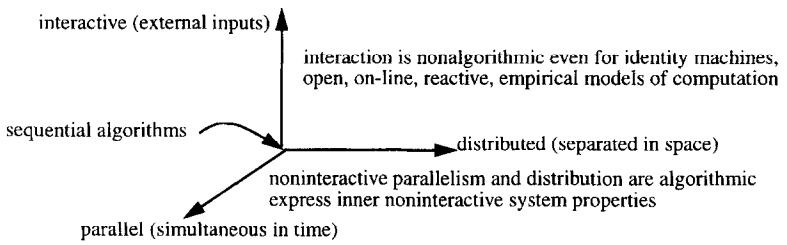


Fig. 7. Design space for interactive, parallel, and distributed computing.

**serializability:** implementation-dependent interface with a scheduler, global condition
operational, implementation-dependent semantics of correctness
**atomicity:** implementation-independent interface with a client, local constraint
applicable to nested and typed transaction models, broader notion of equivalence
**noninteractiveness:** interaction constraint, implies incremental algorithmicity
can be incrementally relaxed to broader notion of correctness allowing controlled interaction

Fig. 8. Transactions correctness as an interaction constraint.

The correctness condition for transactions was first expressed in terms of serializability (equivalence to a serial computation). Lynch [12] suggested that serializability was needlessly implementation-dependent and that atomicity provided a higher-level correctness condition with certain technical advantages, focusing on local rather than global properties of components. Atomicity bridges the gap between state-transition and interaction models, since it can be described either by uninterruptability of state-transition actions or by noninteractive granularity of observation.

"Local noninteractiveness," which views transactions as piecewise noninteractive locally algorithmic segments, presents transaction correctness in a new light as a condition for noninteractive and therefore locally algorithmic behavior. Its outward-directed focus on absence of interaction goes further than atomicity in freeing itself from dependence on state-transition semantics. Viewing transaction correctness as *interaction control*, expressed by absence of interaction rather then atomicity of state transitions, suggests models for transaction correctness different from that of concurrency control.

**O21 (correctness conditions):** serializability → atomicity → local noninteraction

Local noninteractiveness directly implies local algorithmicity, providing an immediate rationale for this correctness condition that invites extension to limited interactiveness that preserves locally algorithmic behavior. In contrast, limited atomicity is inconsistent: atomicity cannot be relaxed incrementally. It is easier to manage and monitor controlled interactiveness than controlled nonatomicity (see Fig. 8).

Applications like collaborative text editing that violate transaction atomicity are more naturally expressed by constraints on interaction protocols (interaction control) than by constraints on concurrency of state transitions (concurrency control).

**P21 (transactions):** Interactive correctness conditions are more natural than serializability.

## 10. Process models: Elements of interaction, unicasting

Milner's seminal paper on process models [14] anticipated that functions cannot express meanings of processes and that histories play a role in specifying the semantics of concurrency. Processes expressed by "P → a.P" are streams with an initial action a and future history P. Alternating input and output actions are given by "stream → input.output.stream" or by the interaction grammar "game → player2.player1.game", which describes game streams in which the opponent makes the first move.

Process models [15] specify the semantics of a single act of interaction by substitution of an argument for a variable, as in lambda calculus reduction:

*reduction rule*: *lambda*$(x)M$ *applied to* $N$ → *substitute* $N$ *for* $x$ *in* $M$ → *send* $N$ *to receive*$(x)M$

In Milner's calculus for communicating systems (CCS), and its later refinement the pi calculus [15], the receive command binds values dispatched by a send command by extending intra-process to inter-process substitution. The connection between a receiving process P and a sending process Q, indicated in the lambda calculus by textual proximity, is specified by a channel name, say n:

*interactive reduction rule*: $n\_receive(x)M.P \mid n\_send(N).Q$ → (*subst* $N$ *for* $x$ *in* $M$).$P \mid Q$

Though the substitution effect of sending a value from a source to a destination is the same between as within processes, this semantics fails to model the control structure for establishing communication channels. Channels are established by a nondeterministic "broadcasting protocol" that dynamically binds senders to eligible receivers at message transmission time, thereby facilitating process mobility.

We call broadcasting followed by committed rendezvous between a sender and receiver "unicasting". The restriction that only a single receiver gets the broadcast message, which differs from the permissive receipt of messages by all receivers in radio and television broadcasting, views broadcast messages as nonreusable entities consumed by receivers. Unicasting is a method of advertising a sales commitment to a single purchaser that invites matching by a symmetric commitment of a buyer. Unicasting ensures nonreusability of messages, interactive commitment characteristic of open systems, and irreversibility of time.

**O22 (CCS):** CCS models computing by algorithmic reduction and interactive unicasting.

The "uni" of unicasting refers to commitment to a unique communication action rather than to a unique receiver. An extension to multi-party communication that preserves the uniqueness of commitment preserves the essence of unicasting. From a semantic viewpoint, unicasting specializes the "." operator to symmetrically triggered input actions and complementary output actions. Both the sender and receiver broad-

cast their availability for communication, symmetrically narrowing their choice from a set of offered alternatives to a specific commitment. Irreversibility becomes a property of the act of communication that applies symmetrically to both senders and receivers rather than an asymmetrical property of input actions.

Unicasting is an interdisciplinary interaction paradigm for incremental narrowing of possible to actual worlds that reflects the asymmetry of time. It is a pervasive mechanism in nature that triggers biological, chemical and computational interaction. It models biological binding in DNA, where affinity between complementary pairs A-T and C-G determines binding of DNA sequences [27]. It models biological coupling between males and females where each partner broadcasts promiscuous availability but makes a commitment to just a single partner, as well as reproductive coupling between sperm and eggs. Chemical binding of positive and negative ions is likewise realized by unicasting to all ions of opposite polarity.

**O23 (unicasting):** Unicasting is a robust communication primitive in both real and artificial worlds.

**P22 (unicasting):** Unicasting models computational, chemical, biological, and sexual interaction.

Living organisms are defined in biology textbooks by the property of interacting with their environment [20]. Proteins and nucleic acids (DNA, RNA) are structured to support chains of interactive elements at the molecular level, using the idea of "backbones" whose role in providing hooks for interaction is similar to that of network backbones. Reproduction is an extremely complex process of interactive matching among several billion base-pairs of a chromosome chain. Interaction requirements dominate function requirements in determining the form of chromosomes and other biological structures. More generally, software architectures for both application-independent software engineering and application-dependent domains like biology have characteristic interaction patterns determined by interaction requirements.

**O24 (architectures):** Network and chromosome architectures have "backbone" interaction patterns.

Viewing biological and other domains in terms of their interaction patterns rather than state-transition rules can provide useful qualitative insights and new forms of abstraction for computational models. Biological problems of alignment and protein folding conform to interaction constraints that identify algorithmic regularities in a nonalgorithmic space of interactive possibilities.

The pi calculus generalizes CCS by transmitting names rather than values across send-receive channels, while preserving unicasting as the control structure for communication. Since variables may be channel names, computation can change the channel topology and process mobility is supported. The unicasting protocol for binding names captures the semantics of mobile processes and of server processes whose ports are bound to senders at message-receiving time.

The pi calculus is a coordination calculus that aims to express "who" communicates by extending algebraic laws developed to express "what" is communicated. Focusing on the problem of "who" processes communicate with turns out to allow the question

```
action terms: A -> x_send(z).P        -- send z on channel x, then do P
               x_receive(y).P         -- substitute received name for y in P, then do P
      terms: P -> A1 + A2 + ... An    -- alternative (nondeterministic) actions
               P1|P2                  -- interactive composition
               hide(y)P               -- restriction (hiding)
               !P                     -- replication (arbitrary number of compositions)
```

Fig. 9. Syntax of the pi calculus.

of "what" is communicated to be handled as a tractable subproblem. However, calculi for names have an imperfectly understood interactive semantics while calculi for values have a well-understood algorithmic semantics.

**P23 (names):** The pi calculus for names is an interactive analog of the CCS calculus for values.

The pi calculus is more complex than CCS because it contains rules for nondeterminism, information hiding, and replication with no analog in the lambda calculus. Its syntax may be defined as in Fig. 9.

The basic computation step (message passing by substitution of values (names) for the bound variable y in the receiver) is the analog of the substitution of arguments for bound variables in the lambda calculus. However, the control structure that determines when interactive steps are executed has an elusive and inherently nondeterministic semantics. The similarity between lambda and pi calculus computing steps represents the tip of an iceberg whose submerged computing engine must cope with broadcasting, hiding, nondeterminism and many other semantic communication issues entirely absent from the lambda calculus.

**O25 (pi calculus):** The pi calculus expresses interactive, nonalgorithmic behavior of process models.

Milner's recognition of the importance of identifying the "elements of interaction" and his pursuit of this goal by extension of the lambda calculus provide remarkable insights into the foundations of interaction but are a beginning rather than a complete foundation for interactive computing. Though the simplicity and power of the pi calculus is aesthetically appealing, its primitives are too low-level to specify real interactive applications, just as lambda calculus primitives are too low-level for algorithmic applications.

Calculi based on interleaving semantics like the pi calculus are limited in their expressiveness because they do not model true concurrency. Though interleaving models are more powerful than algorithmic models, they are less expressive than true concurrency. Interleaving models can be expressed by traces, while nonserializable systems are expressible only by more powerful nonserializable histories.

Both the pi calculus and interaction machines extend the Church–Turing model to interaction, but the pi calculus extends the lambda calculus while interaction machines extend Turing machines. The equivalence of Church and Turing models for algorithms does not extend to interactive computation because nonserializable nonatomic operations with duration cannot be expressed by lambda calculus extensions based on interactive reduction and unicasting that yield trace-based interleaving models of

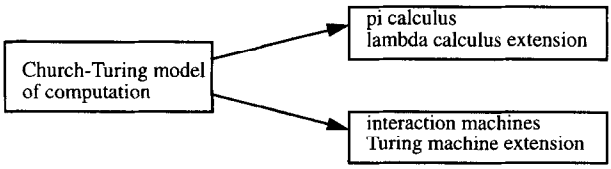


Fig. 10. Interactive extensions of the Church–Turing model.

semantics. The impact of state on expressiveness is greater when state can persist over the life cycle of a systems and is sharable among nonatomic operations having duration than when state is local to a single execution of an algorithm (a single interaction). Multiple processes with persistent shared state can express forms of interactive concurrency not expressible by stateless models that realize interleaving concurrency (see Fig. 10).

**P24 (pi calculus):** The pi calculus has the expressive power of serializable interaction machines.

The pi calculus expresses interleaving semantics but not nonserializable true concurrency. Our definition of expressiveness for sequential interaction machines by interleaving semantics, bisimulation, and game semantics suggests that this robust criterion of expressiveness also covers the pi calculus.

**O26 (shared state):** Interaction machines realize more powerful interaction than the pi calculus.

## 11. Asynchronous and nonserializable interaction: Physics as interactive computation

Since models of physics and computation differ only in the inner structure of components, it is not surprising that they have related models of interaction. Newtonian, relativistic, and chaos/quantum models of physics have models of interaction related to synchronous, asynchronous, and nonserializable computing.

Synchronous versus asynchronous interaction distinguishes systems on the basis of global (Newtonian) time versus relativistic time. Sequential versus nonserializable interaction distinguishes systems along a different dimension related to physical models of chaos: both model sensitivity to initial conditions.

Synchronous systems have a global notion of time like SIMD and MIMD systems, while asynchronous systems have a local notion of time associated with each software component but no global notion of time. Note that synchronous versus asynchronous message passing is a weaker notion that provides local synchrony between senders and receivers but falls short of the global synchrony of a synchronous system, which requires synchronous execution of all instructions.

**P25 (asynchrony):** Asynchronous is more expressive than synchronous interaction.

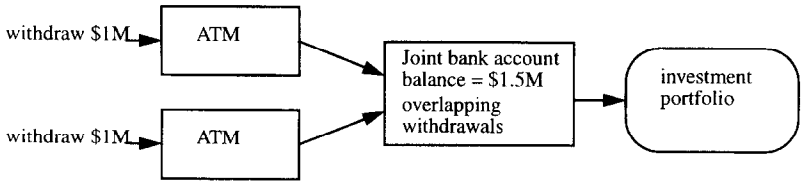


Fig. 11. ATM system as an asynchronous interaction machine.

The behavior of distributed systems with no global notion of time clearly includes synchronous systems with global time as a special case. Distributed systems with autonomous components have no notion of global time, though they can agree on a common approximate time by clock-synchronization algorithms. Interaction histories of asynchronous systems are richer than those of synchronous systems.

Synchronous adversaries control "what" inputs an agent receives, while asynchronous adversaries additionally control "when" an input is received. Asynchronous adversaries who can decide when to zap you are interactively more powerful than synchronous adversaries who merely control content and not time.

Nonserializable interaction, illustrated by the familiar example of a joint bank account, is conceptually very different from asynchronous interaction. Suppose that a joint bank account contains $1.5 million and that two clients simultaneously try to withdraw $1 million at different ATMs. Assume that the withdrawal process requires adjusting an investment portfolio and is therefore not instantaneous (see Fig. 11). A transaction system could in principle handle this situation by satisfying only one client and aborting the transaction of the other. But concurrent interactive systems cannot be presumed to be transactionally well behaved: we must model and manage breakdown of transactional behavior, just as psychologists must model and manage nervous breakdown in people, and physical systems must cope with chaos.

Transactions are a computational mechanism for handling system overload caused by multiple simultaneous demands on a resource. The effect of concurrent potentially conflicting operations op1, op2 of an object can, in the absence of transaction atomicity, be arbitrary and chaotic. Behavior becomes *nonserializable* in that it does not correspond to any sequential execution of the operations op1 and op2. Nonserializability of concurrently executed operations of an object's interface specializes nonserializability of database transactions by considering only atomicity of interface operations, but is essentially similar.

Though nonserializable behavior is considered undesirable in many contexts, it is more expressive (observably richer) than serializable behavior and can be harnessed for useful purposes. Aborting a transaction or replacing a time-consuming optimal algorithm by an approximate just-in-time algorithm is a technique for managing nonserializable behavior. Aborted transactions and just-in-time functions replace ideally desired functionality by less desirable functionality that can be serializably realized.

**P26 (nonserializability):** Nonserializable is more expressive than serializable interaction.

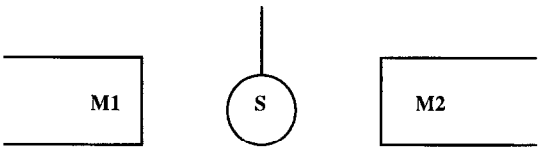


Fig. 12. Chaotic two-dimensional motion of steel pendulum in a magnetic field.

*aborting transactions to achieve serializability may eliminate desirable expressiveness*

Transaction management systems that guarantee atomicity are safe but often too conservative. Permiting some degree of controllable nonserializability, just like permitting some unsound behavior in tasks like error checking, can be worthwhile. For example, optimistic concurrency-control systems lower their guard on the assumption that no conflicts occur and pay for this by requiring drastic and time-consuming actions when this assumption is violated. High-achieving people, like efficient computers, operate close to the margins of nonserializability and pay the price of greater stress and a higher incidence of nervous breakdowns than person opting for comfort at the expense of achievement.

**O27 (nonserializability):** Nonserializability enhances expressiveness but compromises safety.

Nonserializability causes unobservable and uncontrollable temporal sensitivity in accessing a shared resource (the object's state) that is analogous to chaos in physics: sensitivity among competing clients to shared computing resources corresponds to sensitivity among competing forces on a shared physical object. The pattern of competing access to shared resources in the bank account example arises in the well-known demonstration of physical chaotic behavior of Fig. 12, which illustrates a two-dimensional pendulum (steel ball) in the presence of two magnets. The magnets M1, M2 correspond to the operations op1, op2, while the steel ball corresponds to the state. The magnets exert two streams of impulses on the steel ball, just as operations exert streams of impulses on the object's state.

Chaotic behavior in physics is modeled by nonlinear differential equations. The differential equations for a pendulum have a linear first-order behavior but non-linear second-order terms when the first-order terms cancel each other. Pendulum behavior is linear when in the force field of one of the magnets but non-linear (chaotic) in regions where the force fields cancel each other out.

Nonserializability combined with information hiding gives rise to inherent nondeterminism analogous to the nondeterminism of quantum theory. Interference of light passing through two slits on a screen has the same interference structure as the examples of magnets and bank accounts, with slits playing the role of operations (magnets) and the screen serving as the shared state. Though the inner details of behavior for quantum theory and chaos are very different, the interference between competing attempts to access a shared resource is similar, giving rise to similarity in the style of associated computational models.

The boundary between Turing and interaction machines corresponds to that between closed and open subsystems, while that between synchronous and asynchronous machines corresponds to that between Newtonian models with absolute time and relativistic models whose frames of reference have local time. Though relativistic effects are negligible for distributed systems (unless distributed across spaceships), the principle that time is relative to a frame of reference with a subjective notion of simultaneity and an objective notion of causality applies to both physical and computational objects.

Insights derived from interactive models of computation could well prove useful in the study of foundations of physics, since incompleteness, asynchrony, nonserializability, and abstraction effects of interaction are similar in physics and computing. Incompleteness distinguishes between empirical and formal models and asynchrony distinguishes between synchronous and asynchronous models, while nonserializability is the cause of interference effects responsible for the phenomena of both chaos and quantum theory.

**O28 (physics):** Interactive models of computing are strongly related to models of physics.

*incompleteness: distinguishes empirical interactive models from formal algorithmic models*

*asynchrony: distinguishes models with universal time from frames of reference with relative time*

*nonserializability: expresses interference among competing interactions, characteristic of chaos*

*abstraction: hiding of inner actions causes nondeterminism, granularity causes quantum effects*

The nondeterminism of quantum theory is due to abstraction effects that make "chaotic" interference inherently unobservable. Quantum effects in physical theories arise because abstractions have a minimal granularity imposed by limitations of observability. They do not occur in systems whose granularity of abstraction is defined by the designer rather than by physical laws, but distributed models whose observation granularity is system-defined could in principle give rise to computational quanta.

**P27 (physics):** Distinctions among Newtonian, relativistic, chaos, and quantum-theory models of physics can be characterized by styles of interactive computing.

The often-asked question of whether the universe is deterministic can be expressed in terms of the question, "Is the universe an open or a closed system?" In the real world no system is entirely deterministic since it is open to disturbance from an external environment, but we can for practical purposes study systems like the solar system as closed deterministic systems. When this question is asked about the universe, it becomes a question about whether the universe is closed or open in the topological sense of containing its limit points. If it is topologically closed it has a chance of being deterministic, while if it is topologically open it cannot be deterministic because it is subject to unpredictable external forces.

**O29 (openness):** Open computing models provide an interdisciplinary basis for expressing interaction.

*open mathematical space: geometrical region that does not include its external limit points*
*open physical model: physical system subject to external forces*
*open interactive model: computing system subject to external inputs*
*open society: society whose norms can evolve as a result of experience*

**P28 (openness):** Openness in mathematics, physics, and computing has a common foundation.

## PART III : INCOMPLETENESS AND ROBUSTNESS

Incompleteness is viewed as a positive attribute of interactive systems necessary for expressiveness rather than as a negative obstacle to formalization. Interactive expressiveness provides a robust basis for a broad class of models that parallels the robustness of Turing machines. The bipolar robustness of algorithmic and interactive models corresponds to the distinction between closed and open systems, programming in the large and programming in the small, rationalism and empiricism, etc.

### 12. Incompleteness of interactive models: Limitations of logic, program correctness

Incompleteness implies that reducing systems to logic is not merely overambitious but inherently unachievable. The goals of research on formal methods must be modified to reflect that completely proving interactive correctness is not merely impractical but actually impossible. For example, the goals of the fifth-generation computing project of expressing interactive systems by logic are not merely hard to realize but unachievable in their pure form [24].

**O30 (proofs):** All proofs are expressible by programs, but not all programs have correctness proofs.

**P29 (logic):** Logic is too weak to model interactive computation.

Formal reasoning, like algorithmic computing, is a noninteractive step-by-step process from a starting point to a result. Though rules of inference are chosen nondeterministically while algorithm execution is deterministic, this tactical difference of control does not give rise to strategic differences of expressiveness. Both algorithms and proofs are closed systems that exclude interaction during problem solving:

**O31 (mapping logic into computation):**
*logical system → programming language*
*well-formed formulae → programs*
*theorem to be proved → initial input*
*rules of inference → nondeterministic rules of computation*
*proofs → sequential algorithmic computations*

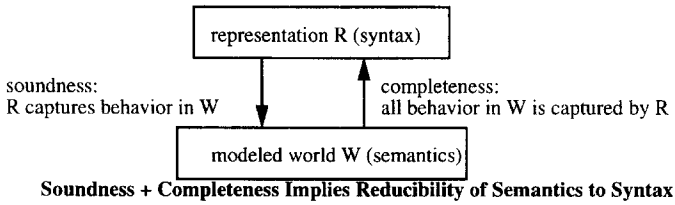


Fig. 13. Relations between syntax and semantics in logical models.

Models in logic and computation aim to capture semantic properties of a modeled world by syntactic representations for the pragmatic benefit of users. Interactive models may have multiple pragmatic modes of use (interfaces), while logics have a single pragmatic interpretation determined by the syntax.

**D16 (models):** A model M = (R, W, I) expresses representations R of modeled worlds W interpreted by human or mechanical interpreters I. R, W, I specify the syntax, semantics, and pragmatics of the model.

This definition extends logical and algorithmic models specified entirely by syntax and semantics, adding a pragmatic component that expresses the role of environments, observers, users, and interpreters [30].

Logical formulae are interpreted as true/false assertions about a *modeled world* of functions and predicates. Formulae true in all models (interpretations) are called *tautologies*. Theorem proving expresses reasoning about tautologies by syntactic rules for proving whether or not a formula is derivable by rules of inference from axioms.

**D17:** A logic is *sound* if all provable formulae are tautologies, *complete* if all tautologies are provable.

Soundness and completeness relate syntactic representations R to their semantic modeled worlds W, though they capture only properties true in all modeled worlds and have little to say about properties of specific modeled worlds. Soundness ensures that representations correctly model behavior of their modeled worlds, while completeness ensures that all possible behavior is modeled. Soundness and completeness together ensure that a representation correctly captures all behavior in the world being modeled. But completeness restricts semantics to closed modeled worlds completely expressible by a representation independently of external (empirical) influences. It constrains modeling power to syntactically expressible behavior (see Fig. 13).

Soundness ensures that syntactic proofs are semantically correct, while completeness ensures that all semantic meaning is syntactically expressible: completeness measures the comprehensiveness of the proof system in expressing semantic meaning. Soundness and completeness together imply that W is reducible to R (the semantic world W is equivalent to and no richer than its representation R). Reducibility of W to R implies completeness of R in expressing W, while incompleteness implies irreducibility.

Though first-order logics have an uncountable number of models, the number of theorems provable from axioms is recursively enumerable. If the logic is both sound and complete, then there is a one-to-one correspondence between syntactic theorems

and semantically true assertions for all models, and the number of true assertions expressible by theorems is recursively enumerable.

**P30 (models):** Sound and complete models have an enumerable number of true statements.

Godel proved incompleteness using a diagonalization argument to show that true statements were not recursively enumerable [10]. Incompleteness of interaction machines follows from the stronger property that the set of computations are not enumerable. The incompleteness proof for interaction machines is actually simpler than Godel's, following directly from nonenumerability of infinite sequences. Since interaction machines are more strongly incomplete than integers, incompleteness is easier to show.

**P31 (incompleteness):** Interaction machines have no sound and complete first-order logic.

**O32:** Irreducible, noncompositional, open, empirical, or interactive systems are necessarily incomplete.

Incompleteness of interaction machines is strongly related to both noncompositionality and emergent behavior. All three concepts are simply alternative ways of describing system behavior for which the whole is greater than the sum of its parts. Compositionality allows the whole to be completely specified in terms of the sum of its parts and therefore implies completeness, while noncompositionality means that the whole cannot be completely specified by its parts. Emergent behavior is behavior that emerges noncompositionally from component behavior and cannot be completely captured by compositional formal systems.

Godel's incompleteness result for arithmetic over the integers is a particular case of a broad class of incompleteness results that brings out the fundamental limitations of completeness. Incompleteness is a necessary price to pay for modeling independent domains of discourse whose semantic properties are richer than the syntactic notation by which they are modeled. Completeness is possible only for a restricted class of relatively trivial logics over semantic domains reducible to syntax. It restricts behavior to that describable by algorithmic proof rules. Models of the real world and even of the integers sacrifice completeness in order to express autonomous (external) meanings.

Complete describability, compositionality, and nonemergence of new behavior are seen to be equivalent restrictions on expressiveness. The converse properties of incomplete describability, noncompositionality, and emergent behavior are equivalent characterizations of unformalizable expressiveness. Computing goes beyond logic in providing systematic techniques for dealing with unformalizable systems.

**O33 (incompleteness):** Incomplete systems express richer behavior than complete systems.

*complete = compositional = no emergent behavior = formalizable*

*incomplete = noncompositional = emergent behavior = unformalizable*

Logics that find errors in programs illustrate that soundness and completeness, though well defined, are often abandoned for practical reasons. Error-finding logics are sound

if they generate error messages only when the program has an error and complete if they discover all errors:

**D18 (error-checking logics):**

*soundness*: *if error message then error*

*completeness*: *if error then error message*

**P32 (errors):** Systems for finding errors in programs are neither sound nor complete.

Though soundness and completeness are well defined by the condition that error messages occur if and only if there is an error, neither is practically useful. Sound logics conservatively exclude useful error messages for which soundness cannot be guaranteed, while complete logics recklessly generate many spurious (unsound) error messages. Practical systems are neither sound nor complete, generating some erroneous messages and missing some errors to strike a balance between caution and aggressiveness.

The reasons for abandoning soundness and completeness in this domain bear further analysis. Our goal is to check that a syntactically defined error-detection system captures an independent ssemantic notion of error. Since the semantic notion cannot be precisely defined it cannot be completely formalized, but the semantic notion of error can be syntactically approximated. In choosing our approximation we avoid the extreme conservatism of soundness and the extreme permissiveness of completeness by compromising (in both the good and bad senses of the word) between conservatism and permissiveness.

**O34 (type 1 and type 2 correctness):** Proofs of existence of correct behavior (type-1 correctness) are generally easier than proofs of the nonexistence of incorrect behavior (type-2 correctness):

*type-1 correctness*: *prove existence of desired behavior*, *local soundness property*, *often provable*

*type-2 correctness*: *prove nonexistence of incorrect behavior*, *global completeness*, *rarely provable*

Insisting on type-1 correctness in all contingencies (soundness) is too conservative, since the existence of correct behavior in favorable circumstances is sufficient. Insisting that all error messages of an error-detection system necessarily correspond to errors is too conservative, while insisting that airline reservation systems work correctly in all possible contingencies (including power failures) is too expensive.

The study of incomplete systems has been strongly resisted on the grounds that we should not attempt to analyze, much less build, systems whose behavior we cannot prove correct. However, practical techniques for design and implementation rarely guarantee correctness. We live in a world where correctness cannot be guaranteed and use ad-hoc testing heuristics to increase the probability of correctness. Guaranteed correctness can rarely be unachieved for real applications. The distinction between algorithmic and interactive systems corresponds to that between toy and real systems and also to that between systems with guaranteed and probable correctness.

Result checking of behavior after it occurs is an important practical correctness technique in cases where it is applicable. Techniques for systematic on-line result

checking will become an increasingly important practical supplement to off-line testing and verification [2]. Since getting the right answer cannot be algorithmically guaranteed by a priori correctness proofs, a posteriori methods of interactive checking for rightness of the answer become increasingly important.

**O35 (result checking):** Interactive (just-in-time) correctness checking will become important.

Result checking that proves properties of computer-generated "theorems" differs from traditional interactive theorem proving that uses interaction to prove a statically determined result. The extension of theorem proving to interactively generated evolving results is the formal analog of extending algorithms to interaction. Proving that computed data has certain properties extends the notion of theorems to moving targets and allows interactive focusing on what actually occurs rather than on sets of all possible events.

**O36:** Proving interactive theorems extends interactive theorem proving.

Formal logic studies processes of inference guaranteed to be valid because of their logical form independently of their subject matter (formal logic → logical form). Formal logic is explicitly nonempirical: it is concerned with "laws of thought" true independently of the empirical propositions being reasoned about. Since techniques of problem solving and model building generally depend on domain-specific properties not expressible by logic, formal logic cannot express general domain-specific modeling.

**O37 (domains):** Formal logic models logical form, while interaction models nonlogical content.

## 13. Robustness of interactive models: Programming in the large and empirical computer science

The robustness of Turing machines in expressing algorithms, functions, and logic is paralleled by an equal robustness of interaction machines in expressing software systems, AI agents, and empirical models. Each left-hand-side concept of Fig. 14 is more expressive than the corresponding right-hand-side concept. Moreover, left-hand-side concepts can be uniformly modeled by a *universal interaction machine*, just as right-hand-side concepts can be uniformly modeled by a Turing machine (*universal algorithm machine*). Interaction machines define a robust notion of expressiveness for left-hand-side concepts just as Turing machines provide robust expressiveness for right-hand-side concepts.

**O38 (robustness):** Software systems, AI agents, and open systems have the same expressive power.

**P33 (robustness):** Interaction has many alternative models with the same expressive power.

The greater expressiveness of interactive over algorithmic computing has been extensively explored. We briefly examine each of the other dichotomies of Fig. 14, defining terms where necessary.

*open systems > closed systems*

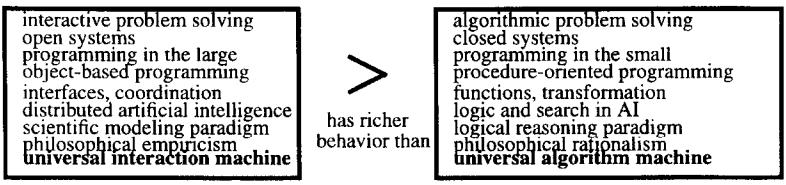


Fig. 14. Robust expressiveness of interaction and Turing machines.

A computing system is said to be open if its behavior during the process of computation depends on external information and closed otherwise. Turing machines and algorithms with inputs are *closed* (their actions do not depend on external interaction), while interactive systems are *open*. Mathematically, open systems have free variables while closed systems have only bound variables.

*programming in the large > programming in the small*

Programming in the small (PIS) is algorithmic, while programming in the large (PIL) is interactive. A sequence of a million arithmetic operations is not PIL, while medium-size embedded software systems are. PIL systems are necessarily interactive but not necessarily large. PIL is not simply scaled-up PIS; it has qualitatively different program structures and models of computation. The irreducibility of interaction to algorithms implies inexpressibility of PIL by PIS. Scaling up shifts attention from inner activities within components to interaction among components. PIL was observed to differ from PIS as early as the 1960s, but the difference was viewed as a quantitative change of scale. Expressing the difference as a qualitative change of expressiveness explains the observed inability to scale up from algorithms to software systems.

**P34 (systems):** Software engineering systems have interactive, nonalgorithmic models.

*object-based programming > procedure-oriented programming*

Interaction machines model objects while Turing machines model functions and procedures. Interaction machines provide a unifying model for objects in software engineering and agents in AI.

*interfaces, coordination > functions, transformation*

Interfaces plays a role in system specification analogous to that played by functions in specifying algorithms. Coordination behavior is the interactive analog of transformation behavior. Semantics of an interactive system is specified by all possible coordination behaviors, just as semantics of an algorithm is specified by all possible transformation behaviors. Coordination is concerned with constraining nonalgorithmic interaction so it can be managed and harnessed for useful purposes.

*distributed artificial intelligence (agent-oriented programming) > logic-based AI*

The paradigm shift in AI from logic and search to interactive models is not merely a tactical change but is a strategic paradigm shift from closed algorithmic to more expressive interactive models. The reasoning/interaction dichotomy is precisely that between good old-fashioned AI and "modern" agent-oriented AI. This paradigm shift is evident not only in research, but also in textbooks that systematically reformulate AI

in terms of intelligent agents [19]. In AI just as in software engineering the transition from toy problems to practical applications involves a shift from purely algorithmic to interactive models.

**O39 (paradigm shift):** The algorithm/interaction paradigm shift permeates all areas of computing.

*scientific modeling paradigm > logical reasoning paradigm*

*philosophical empiricism > philosophical rationalism*

The dichotomy of algorithmic versus interactive models extends to rationalist versus empirical models in physics and philosophy. Interactive models are the computational analog of the 17th-century liberation of the natural sciences from the Platonic world view. The extension from synchronous to asynchronous and nonserializable interaction corresponds to that from the Newton/Laplace model as a synchronous deterministic clock and the relativity/quantum/chaos model as an asynchronous nondeterministic system. Relativity replaces the synchronous Newtonian model by asynchronous observers with independent frames of reference, while quantum theory and chaos reflect nondeterminism and sensitivity to initial conditions:

*Plato/Descartes models → Newton/Laplace models → relativity/quantum/chaos theory*

*algorithmic models → synchronous interactive models → asynchronous/nonserializable models*

The paradigm shift from algorithmic to interactive (empirical) models occurs in many disciplines as they mature. Computer science provides a normative interdisciplinary framework for better understanding the conceptual foundations of such paradigm shifts. It is a lingua franca for modeling that allows common features of interactive models in a variety of disciplines to be uniformly expressed.

The philosophical intuition that empirical models are more expressive than rationalist models can be precisely stated and proved in computational terms. In expressing rationalist versus empiricist models by "algorithms versus interaction" we reduce fuzzy philosophical distinctions to crisp concepts of computation, showing that empiricism is in a precise sense more expressive than rationalism.

**P35 (empiricism):** The intuition that empiricism extends rationalism can be proved for computing.

Irreducibility establishes the intellectual legitimacy of empirical computer science by freeing researchers from the obligation of expressing their models in algorithmic terms. It establishes computer science as a discipline distinct from mathematics and, by clarifying the nature of empirical models of computation, provides a technical rationale for calling computer science a science.

**P36 (empirical CS):** Interaction machines precisely characterize empirical computer science.

The paradigm shift from algorithms to interaction has sparked Kuhnian debates between proponents of algorithmic theory and interactive practice concerning both intellectual legitimacy and funding. Though limitations on the role of formalism appear to provide ammunition against theoretical research and imply that certain kinds of

theoretical research such as attempts to reduce object-based to functional or logic programming should be abandoned, better understanding of the role of theory can in fact refocus theoretical research to support practice more effectively, just as it has done in the natural sciences.

**O40 (research):** Models of interaction can refocus research to serve practice more effectively.

Computing technology has undergone a paradigm shift from complete mathematical models to incomplete but more expressive interactive models. Escape from the Turing tarpit into a new interactive dimension enriches the design space of models of computing so that applications can be more effectively expressed. Moreover, focusing on interaction patterns can often provide useful qualitative insights and new abstraction frameworks. Domain-specific models of biology as well as domain-independent models of software engineering take on new meaning when components are viewed in terms of how they interact rather than in terms of what they compute [28, 29]. Interaction machines express the paradigm shift from algorithms, logic, and mathematics to empirical interactive models in a natural and simple form.

## Appendix: List of propositions

This list of propositions provides a profile of differences between algorithmic and interactive models. Many are simply alternative statements of the thesis that interaction is more expressive than algorithms. Collectively these propositions show that the interactive computing paradigm requires traditional assumptions about the nature of computing to be fundamentally revised.

### Propositions

**P1 (Turing machines):** TMs cannot model interaction since they shut out the world while computing.

**P2 (interaction machines):** Interaction machines cannot be modeled by Turing machines.

**P3 (nonenumerability):** The interaction histories of an interaction machine are nonenumerable.

**P4 (on-line algorithms):** On-line processes with the closed-system property are on-line algorithms.

**P5 (complexity):** Under suitable locality conditions, problems with algorithmic complexity NP have interactive complexity P.

**P6 (dynamic interaction):** Dynamic interaction is more expressive than on-line algorithms.

**P7 (constraints):** Constraints can specify nonalgorithmic noncompositional emergent behavior.

**P8 (grammars):** Interactive listening machines can express richer behavior than generative grammars.

**P9 (inclusion):** Dynamic inclusion refines set inclusion as a measure of expressive power.

**P10 (expressiveness):** Bisimulation, dynamic inclusion, and game semantics are equally expressive.

**P11 (irreducibility):** Extensional behavior cannot express intensional behavior and vice versa.

**P12 (concurrency):** Interleaving models, enhanced operational semantic models, and true concurrency have progressively greater expressive power.

**P13 (nonmonotonicity):** Openness and interactiveness are nonmonotonic system properties.

**P14 (duality):** Observation/control duality in control theory mirrors algorithm/interaction duality.

**P15 (noncompositionality):** Behavior of processes and persistent components is not compositional.

**P16 (frameworks):** Frameworks can be specified by constraints on constituent components.

**P17 (identity):** Interactive identity machines can express richer behavior than Turing machines.

**P18 (agents):** Agents interacting with nonalgorithmic systems have nonalgorithmic behavior.

**P19 (management):** Interactive management is more expressive than rule-based management.

**P20 (orthogonality):** Interaction, parallelism, and distribution are orthogonal forms of behavior.

**P21 (transactions):** Interactive correctness conditions are more natural than serializability.

**P22 (unicasting):** Unicasting models computational, chemical, biological, and sexual interaction.

**P23 (names):** The pi calculus for names is an interactive analog of the CCS calculus for values.

**P24 (pi calculus):** The pi calculus has the expressive power of serializable interaction machines.

**P25 (asynchrony):** Asynchronous is more expressive than synchronous interaction.

**P26 (nonserializability):** Nonserializable is more expressive than serializable interaction.

**P27 (physics):** Distinctions among Newtonian, relativistic, chaos, and quantum-theory models of physics can be characterized by styles of interactive computing.

**P28 (openness):** Openness in mathematics, physics, and computing has a common foundation.

**P29 (logic):** Logic is too weak to model interactive computation.

**P30 (models):** Sound and complete models have an enumerable number of true statements.

**P31 (incompleteness):** Interaction machines have no sound and complete first-order logic.

**P32 (errors):** Systems for finding errors in programs are neither sound nor complete.

**P33 (robustness):** Interaction has many alternative models with the same expressive power.

**P34 (systems):** Software engineering systems have interactive, nonalgorithmic models.

**P35 (empiricism):** The intuition that empiricism extends rationalism can be proved for computing.

**P36 (empirical CS):** Interaction machines precisely characterize empirical computer science.

## References


[1] J. Andreoli, C. Hankin, D. Le Metayer, Coordination Programming: Mechanisms, Models, and Semantics, 1996.
[2] M. Blum, Result Checking, MIT Distinguished Lecture, 1994.
[3] G. Brassard, A Quantum Jump in Computer Science, Lecture Notes in Computer Science 1000, Springer, Berlin, 1995.
[4] P. Ciancarini, Coordination models and languages as software integrators, Comput. Surveys 28 (1996) 300–302.
[5] P. Degano, R. DeNicola, U. Montanari, Universal axioms for bisimulations, Theoret. Comput. Sci. 114 (1993) 63–91.
[6] P. Degano, C. Priami, Enhanced operational semantics, Comput. Surveys 28 (1996) 352–354.
[7] T. Dean, M. Wellman, Planning and Control, Morgan Kaufman, Los Altos, CA, 1991.
[8] E. Dijkstra, Go to considered harmful, CACM 1968.
[9] D. Garlan, Research directions in software architecture, Comput. Surveys 27 (1995) 257–261.
[10] K. Godel, On formally undecidable propositions of Principia Mathematica and related systems, translation of 1931 article in Monatshefte fur Mathematik und Physik, in: M. Davis (Ed.), The Undecidable, Raven Press, New York, 1965.
[11] D. Knuth, The Art of Computer Programming, vol. 1, Addison-Wesley, Reading, MA, 1968.
[12] N. Lynch, Distributed Algorithms, Morgan Kaufman, Los Altos, CA, 1996.
[13] Z. Manna, A. Pnueli, The Temporal Logic of Reactive and Concurrent Systems, Springer, Berlin, 1992.
[14] R. Milner, Processes: a mathematical model of computing agents, in: Logic Colloquium '73, North-Holland, Amsterdam, 1975.
[15] R. Milner, Elements of interaction, CACM 36 (1993) 78–89.
[16] M. Minsky, The Society of Mind, Simon and Schuster, New York, 1986.
[17] C. Papadimitriou, Games against nature, J. Comput. System Sci. 31 (1985) 288–301.
[18] C. Papadimitriou, M. Yannakakis, Shortest paths without a map, Theoret. Comput. Sci. 84 (1991) 127–150.
[19] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Addison-Wesley, Reading, MA, 1994.
[20] J. Setubal, J. Meidanis, Introduction to Computational Molecular Biology, PWS, Boston, MA, 1997.
[21] H. Simon, The Sciences of the Artificial, 2nd ed., MIT Press, New York, 1982.
[22] W. Thomas, Automata on infinite objects, in: J. Van Leeuwen (Ed.), Handbook of Theoretical Computer Science, vol. B, MIT Press, New York, 1990.
[23] A. Turing, Systems of logic based on ordinals, Proc. London Math. Soc., 1939.
[24] P. Wegner, Tradeoffs between reasoning and modeling, in: Agha, Wegner, Yonezawa (Eds.), Research Directions in Concurrent Object-Oriented Programming, MIT Press, New York, 1993.
[25] P. Wegner, Interaction as a basis for empirical computer science, Comput. Surveys 27 (1995) 45–48.
[26] P. Wegner, Interactive foundations of object-based programming, IEEE Computer 28 (1995) 70–72.
[27] P. Wegner, Why interaction is more powerful than algorithms, CACM 40 (1997) 80–91.
[28] P. Wegner, Interactive software technology, in: Handbook of Computer Science and Engineering, CRC Press, Boca Raton, IL, 1996.
[29] P. Wegner, Frameworks for active compound documents, Brown University TR-97-01, January 1997.
[30] P. Wegner, The expressiveness of models, Brown University, Technical Report, August 1997, www.cs.brown.edu/people/pw.