A Research Agenda for Interactive Computing
Peter Wegner, Extended Abstract, January 11 1998

Abstract: This paper presents evidence for the claim that interactive behavior cannot be expressed by Turing machines, examines the impact of this claim on selected subareas of the theory of computation, and proposes an agenda of research goals on interactive computing (G1-G65). We start from the need to extend computational problems from algorithms to embedded and real-time systems. This extended view of computing requires extension of models of computation so they model interactive processes and of expressiveness so it models services over time and external worlds. The intuitive notion of computing must be broadened beyond Church's thesis, and observability must be extended to match the richer notion of interactive behavior. The impact of irreducibility on process models, constraints, testing, nondeterminism, time, on-line algorithms, and open systems is examined, and goals that extend or reinterpret concepts and results in each of these areas are proposed. By escaping from the Turing tarpit we can examine the contours of the interactive design space from an external vantage-point and formulate a detailed agenda of goals for extending models of computation from algorithms to interaction.

1. Extending Fundamental Concepts

We start from the need to extend computational problems from algorithms to embedded and real-time systems [We1]. This extended view of computing requires the extension of models of computation so they model interactive processes and of expressiveness so it models services over time and external worlds. The intuitive notion of computing must be broadened beyond Church's thesis, and observability must be extended to match the richer notion of interactive behavior.

Computational problem: extend from algorithms to reactive, embedded, and time-critical systems

Algorithmic problems like sorting, graph reachability, maximum flow, and the traveling-salesman problem [Pa] express parameterized decision and optimization problems specifiable by computable functions. *Interactive problems* like airline reservation, robot control, and speech recognition provide interactive services over time. Algorithm complexity is specified by a size parameter, while interactive problems like airline reservation vary in size and complexity along many dimensions. Algorithm behavior is specified by computable functions while interactive systems have no complete formal behavior specification [We1].

G1: Characterize "interactive problem" and classify modes of interactive problem solving.

Model of computation: extend from Turing machines to interaction machines

Algorithms are modeled by *Turing machines (TMs)* that noninteractively transform finite input strings to output strings. Interactive computation is modeled by *interaction machines* (IMs), which extend TMs with input and output actions that interact dynamically with the external environment. IMs cannot be modeled by TMs with a finite initial input tape because any finite input stream can always be extended by an adversarial environment [We2]. This "proof" implies that IMs are as powerful as TMs with infinite initial input tapes and model reactive systems [MP] and TMs with oracles [Tu, Pa], which are known to be more expressive than TMs. IMs are here defined intensionally as machines with an effectively checkable "interaction property". They may have single or multiple input streams, synchronous or asynchronous IO actions, and can differ along many other dimensions, but all forms of interaction express open-system behavior. IMs model objects, agents, and actual computers more directly than TMs.

G2: Develop a precise definition of IMs that supports multiple interactive extensions of TMs.

The subclass *sequential interaction machines (SIMs)*, obtained by restricting IMs to a single sequential input stream, expresses reactive systems, objects, and input/output automata [LMWF], but not airline reservation, speech and vision, robots, or VR systems. There is strong evidence [We2] that IMs with multiple streams are more expressive than SIMs. This contrasts with the fact that multiple-tape and single-tape TMs have the same expressiveness. Proving that multiple-stream IMs are more expressive than SIMs requires a precise model of multiple-stream computation and a careful definition of expressiveness.

G3: Precisely define SIMs and show they are less expressive than multi-stream IMs.

Expressiveness: extend computable functions to services over time and models of external worlds

Expressiveness for algorithms is specified by computable functions and recursively enumerable sets. Interactive expressiveness is a nonfunctional notion that can be measured by the ability to provide services over time or model the external world. Though interactive expressiveness for SE, AI, and other domains cannot be explicitly or completely defined, a definition sufficient to show that TMs are less expressive than IMs and that different classes of IMs have different expressiveness can be given. Both algorithms and interactive systems have a declarative semantics that specifies "what" independently of "how". Denotations of algorithms are defined as fixed points of a lattice-structured semantic domain, while denotations of interactive systems are partially specified external entities with no fixed point semantics [We2].

G4: Specify interactive expressiveness, prove TMs are more expressive than IMs, classify IMs.

Intuitive notion of computing: extend Church's thesis to empirical computer science

Church's thesis, broadly interpreted, equates the intuitive notion of computing by algorithms with the formal notion of computability by TMs. However, the interactive expressiveness of embedded reactive systems, agent-oriented programming, and speech recognition cannot be captured by algorithms or TMs. Whereas algorithms adequately express number crunching and isolated computation of outputs from inputs, they do not express social and technical computer infrastructures, which require broadening the intuitive model of computing to include interaction. The broader intuitive notion of computing provides a technical rationale for broadening the agenda for computer science and engineering [HL].

G5: Characterize the intuitive notion of interactive computing and relate it to empirical CS.

Observability: extend off-line to on-line observation and testing

On-line observers that interact with a system while computing are more expressive than off-line observers of noninteractive behavior: they can make finer-granularity observational distinctions. Bisimulation is an on-line mode of observation of state transitions that expresses interaction of two-person games and can be specified by interaction grammars [We2]. The Brock-Ackerman anomaly is an on-line mode of observation of input-output behavior more expressive than off-line modes of observation. In [GW] we show that on-line observers are more expressive than off-line observers for glass-box observers (bisimulation), black box observers (Brock Ackerman), and statistical observers who observe the distribution of outputs. The greater expressiveness of on-line over off-line observers corroborates the claim that behavior of interactive systems is more expressive than algorithms. Behavior distinguishable by on-line but not off-line testers is nonfunctional behavior, since it cannot be observed by functional testers.

G6: Classify on-line observers and the kinds of nonfunctional behaviors they can observe.

G7: Express bisimulation by on-line equivalence of nonterminals of interaction grammars.

Observation equivalence to an "ideal" process is the interactive analog of correctness but cannot be effectively specified or verified. Observation equivalence [Mi1], testing equivalence [DH], model checking [CW2], and result checking [BK] can be viewed as forms of observability. Testing equivalence [DH], which defines observation equivalence for pairs of systems, is the interactive analog of testing. Model checking can be viewed as a form of testing (of finite approximations), while result checking provides greater certainty (later binding) than testing: it checks programs in actual rather than simulated conditions.

G8: Extend observability, testing, and observation equivalence from algorithms to interactive systems.

G9: Extend model-checking and result-checking techniques to interactive behavior.

2. Extending Computational Models and Abstractions

Algorithms are modeled by a variety of equivalent models with distinct pragmatic goals. Machines model algorithms operationally by their computation mechanism, logics relate syntactic notions of inference and proof to semantic notions of modeling, while grammars relate processes of generating and recognizing sets of strings to computation power for functions. The robust equivalence of these very different models of computation is due in large measure to their noninteractiveness:

algorithms and Turing machines noninteractively compute outputs from inputs
first-order logics noninteractively infer theorems from axioms by rules of inference
generative grammars noninteractively generate language strings from a starting symbol

Equivalence is a two-edged property because new models do not increase expressiveness if they are equivalent to already existing ones. The interactive extension of machines, logics, and grammars yields more expressive models that are not reducible to their noninteractive counterparts or to each other:

Turing machines (noninteractive steps) -> interaction machines (interaction steps)

G10: Extend inner algorithm computation to interactive embedded computation.

logic models (noninteractive inference) -> empirical models (interactive external worlds)

G11: Extend models of inference to models of external contexts, behaviors, and environments.

Chomsky grammars (off-line strings) -> interaction patterns (on-line speech, vision, VR)

G12: Extend time-independent off-line string patterns to time-sensitive on-line interaction patterns.

The rather general goals G10 through G12 are refined into more specific goals in sections 3 through 5 below. Extension from algorithms to interaction arises in a variety of other contexts:

G13: Characterize object-based programming as an interactive extension of procedural programming.

G14: Specify programming in the large as an interactive extension of programming in the small.

G15: Contrast agent-oriented, interactive models of AI with earlier logic and search models.

G16: Extend the Turing test from disembodied thought to embodied agents through interaction [We1].

G17: Model fuzzy open-system concepts by precise interaction-machine concepts.

G18: Extend logical closed-world reasoning to scientific modeling through models of interaction.

Each of these goals includes an interactive concept that is richer than a corresponding algorithmic concept. The algorithmic concepts are expressively equivalent, while the interactive concepts vary along many dimensions. Algorithms, closed system computations, procedure-oriented programming, and logic and search in AI are reducible to a common TM computation model, while corresponding extensions cannot be reduced to a common model of extended interactive computation.

G19: Identify common principles of interactive extension, characterize domain-specific differences.

Filling in the details for these extensions is beyond the scope of this paper, but awareness that interaction is more expressive than algorithms provides new perspectives on models of computation. By escaping from the Turing tarpit we can examine its contours from an external vantage-point.

3. Extending Logic to Interaction

By embedding logic models in the broader context of interactive models, we gain insight into the limits of logic in modeling autonomous external worlds. First-order logic (fol) proves theorems from axioms true for all interpretations of nonlogical symbols by noninteractive inference. Interactive models replace complete semantics of fols by weaker abstractions that sacrifice completeness for expressiveness.

G20: Connect logical incompleteness to descriptive incompleteness of modeled worlds..

The nonenumerability of interpretations of fols, which reflects the nonenumerable external meanings of nonlogical symbols, may be contrasted with recursively enumerable (RE) theorems for a fixed interpretation. Logic freezes interpretations prior to inference so that the number of theorems is RE. Sound and complete logics with an RE number of theorems can express only an RE number of true facts:

logic models: nonenumerable number of interpretations -> nonenumerable number of external contexts
each interpretation has an RE number of theorems -> RE number of true statements

G21: Explore late (dynamic) binding of interpretations in logic as a form of interactive modeling.

Fols reason about truth in all interpretations, corresponding to a specific (free-algebra) interpretation. Soundness and completeness mean that formulae are provable iff they are true in all interpretations. But completeness, like equivalence, is a two-edged property that restricts semantics to be isomorphic to and therefore no richer than syntax [We1]. By abandoning completeness, we can model partially known interactive modeled worlds not completely describable by syntax. Incompleteness is a stronger condition than undecidability, implying nonexistence of complete models rather than merely noncomputability. Undecid-

able problems have a fixed-point semantics while incomplete models do not.

G22: Show that undecidability is stonger than incompleteness and examine implications for modeling.

Interaction extends models so that the interpretation of nonlogical symbols evolves during inference to take account of new data. However, interactive discovery of facts negates the fundamental property that true facts always remain true. Nonmonotonic logics [Re] permit reasoning about incompletely described modeled worlds. A closed-world assumption (cwa) can make evolving logics locally monotonic by assigning default values to every ground formula that is not a theorem, and treating open incomplete systems as closed and complete so that traditional inference is possible. The cwa is a widely used technique for making open systems tractable by closing them temporarily. It is used for AI planning [EKW], fixed-point models of inheritance [We4], atomicity of transactions [LMWF], and on-line algorithms [PY]. There is a close relation between logical closure by the cwa and computational closure that excludes interaction.

G23: Connect logical cwa closure and computational closure by temporarily excluding interaction.

Noninteractive logical inference merely makes implicit knowledge explicit without deriving new knowledge. In contrast, interaction assimilates new (emergent) facts not derivable from what is already known. Thus interactive systems have emergent behavior. If theorems are considered nonemergent, then the fuzzy notion of emergent behavior can be precisely defined as interactive behavior. The strong relation between interactive inference, open interpretation, nonmonotonic logic, and emergent behavior shows that interactive extension is a unifying underlying principle that ties these diverse concepts together.

interactive inference = open interpretation = nonmonotonic logic -> emergent behavior

G24: Explore relations among interactive inference, openness, nonmonotonicity, emergent behavior.

Logical models M = (R, W) express the semantics of modeled worlds W by syntactic representations R. Interactive models have the form M=(R,W,P), where P is an additional "pragmatic" component that specifies modes of interaction with the environment or modes of use of the model by clients [We2]. Interactive models have multiple pragmatics P specified by multiple interfaces [We1]. The pragmatics P expresses both the use of a model by external clients and the perception by an agent of the external world. The canonical (free algebra) fol model has a fixed (degenerate) pragmatics P0 and a fixed modeled world W0 of the form M = (R, W, P0) -> (R, W0). Fixing the pragmatics and modeled world reduces interactive models to noninteractive closed systems.

G25: Develop a logical framework for interactive models with multiple pragmatic interfaces.

From the viewpoint of an agent, pragmatics can be modeled formally as a projection of the world W on the input sensors. Stimuli S from an external world W are projections S = P(W) onto input sensors such that the inverse cannot be completely known. From an external viewpoint pragmatics can be modeled by testers that observe a specified subset of system behavior [DH, GW]. Pragmatic specifications of partial information by systems of their context or by testers of systems are a useful tool for modeling interaction.

G26: Model partial information by pragmatic models with noninvertible projections.

Projection of the world on an agent's sensors is expressed by Plato's cave metaphor, which compares humans to cave dwellers who can observe only shadows on the walls of their cave (retina). S is the shadow cast by W on the walls of the agent's cave. However, S can include stereo inputs from multiple sensors (eyes and/or ears) and temporal inputs at successive points of time, and indeed can be a stereo-spatio-temporal interaction pattern more complex than a two-dimensional image on the walls of a cave.

G27: Develop a mathematical multiple-projection model to express Plato's cave metaphor.

Logic and interactive models both aim to be declarative by expressing what is being modeled independently of how it is modeled. But the noninteractiveness of logic causes the "what" to be limited to interpretations of formulae with function and predicate symbols, while the "what" of interactive models can express properties of an external world. Declarative specifications of "what" independently of "how" denote objects in external worlds for interactive models and denote properties of functions or sets.

G28: Examine the relation between algorithmic and interactive declarativeness

4. Extending Grammars to Interaction Patterns

The Chomsky hierarchy determines equivalences between recognition mechanisms (automata) and generating mechanisms (grammars) for strings:

finite automata <--> regular grammars
pushdown automata <--> context-free grammars
linear bounded automata <--> context-sensitive grammars
Turing machines <--> unrestricted grammars (recursively enumerable sets)

The extension of off-line generative grammars to on-line interaction patterns of speech and vision parallels interactive extension of machines and logic. Grammars for speech and vision developed in the 1960s and 1970s do not generalize from toy examples to practical applications because sets of strings generated by noninteractive off-line rules abstract away from the real-time cognitive process of listening. The distinction between on-line streams and off-line strings is well understood by researchers in pattern theory and real-time systems but is not well established at the level of mainstream models of computation.

G29: Distinguish the semantics of on-line streams from that of off-line strings.

String semantics is determined entirely by the sequence of its elements, is time-independent, and is specifiable by computable functions, while the semantics of streams may depend on time and involve nonfunctional dependencies among elements, especially for "nonsequential" machines that overlap the processing of their inputs. The nonfunctional interdependence of stream elements mirrors that of objects whose operation execution may be overlapped (see section 8). The fact that finite streams and interaction histories cannot be specified by recursively enumerable (algorithmic) specifications is a much stronger result than the failure of Turing machines to express computations for infinite input strings:

grammars that generate off-line strings cannot express on-line interaction patterns of speech and vision

G30: Characterize the nonalgorithmic behavior of finite streams.

Finite streams of sequential interaction machines whose requests for service are time-independent and do not interact can be specified by strings. Time-dependent streams can be specified as time-stamped traces, but semantic equivalence between unstamped and stamped input streams is generally unspecifiable [We2]. There are many loose ends in specifying the semantics of streams. Models like [Br4] that make use of McCarthy's ambiguity operator to define fair merging provide a starting point for stream semantics.

G31: Develop a semantics for multiple streams that share a common state, apply to objects and agents.

The work of Marr [Ma] was influential in moving vision research away from purely algorithmic techniques to the study of the interactive context in which vision occurs. In developing semantic models for interaction patterns we will examine research on pattern theory by Grenander [Gr] and Mumford [Mu]. Extension of grammar models to interaction will allow the computer science community to interact more directly with mathematicians and engineers working in the areas of speech and vision.

G32: Relate work on interaction patterns in speech and vision to models of interaction.

Research on the semantics of streams should be supplemented by work on multiple-stream models along the lines of [Ma, Mu], and of multimodal models of VR. Single, stereo, and temporal images express progressively more powerful modes of sensory perception of the environment:

patterns perceived by a single sensor at a single instant of time (photographs)
patterns perceived by multiple spatial and multimodal sensors at a single instance of time (stereo)
temporal patterns that capture evolving environment/image perspectives over time (streams)

G33: Connect research on pattern theory [GR, Mu] to computational models for interaction patterns.

Though photographs can express images of great complexity, stereo images from multiple sensors can further enhance image perception, while patterns in time are a crucial element in both human and computer perception. Pure spatial and temporal models are a baseline for the study of mixed space-time patterns of interaction and of integrated spatio-temporal models of perception.

G34: Develop a space-time model for interpreting multiple images of a shared, evolving, scene.

5. Irreducibility - A Necessary Consequence of Greater Expressiveness

Irreducibility should be viewed as a positive result that allows the expressiveness of models of computation to be extended rather than as a negative impossibility result. The irreducibility of IMs to TMs is reinforced by many other forms of irreducibility. Incompleteness of interactive models to syntactic proofs of first-order logic is a complementary and mutually reinforcing form of irreducibility. Mutual irreducibility among classes of IMs contrasts sharply with reducibility among Turing-equivalent formalisms and implies that the notion "universal machine" does not scale up from algorithms to interaction. The notions of definability, which implies reducibility of a newly defined thing to an already defined set of things, and formalizability, which implies reducibility of what is being formalized, must be reexamined.

G35: Classify different forms of irreducibility and their associated proof techniques.

Mutual irreducibility among IMs: models and modes of interaction are mutually irreducible

Mutual irreducibility among models of interaction includes irreducibility of multiple- to single-stream IMs and irreducibility of asynchronous to synchronous interaction [We2]. It implies that modeling collaboration is harder than modeling sequential interaction by SIMs, that specification by multiple views or abstractions is more powerful than specification by a single abstraction, and that multiple stereo images of a scene provide more information than a single image. Irreducibility due to sharing of an inner state is examined in [We2]. Sharing of external views is examined in [Mu], which precisely characterizes the additional information of a second image in stereo vision and examines the cognitive evidence that humans use a "minimum description length" principle in reducing redundancy by storing only a single image and a difference representation when processing stereo images. Mutual irreducibility is related to the notion that common nouns such as "table" are "natural kinds" which cannot be defined in terms of other nouns.

G36: Characterize the design space of mutually irreducible interaction machines

Incompleteness: IM behavior cannot be captured by (reduced to) sound and complete first-order logic

Godel incompleteness is a strong result that invalidates not only Russell's and Hilbert's attempts to reduce mathematics to logic but also the reduction of interaction to algorithms. It provides independent (denotational) evidence that supplements and strengthens direct (operational) arguments for irreducibility of IMs to TMs. This does not release us from the responsibility of finding a good formal proof, but it does suggest that mutually supportive cumulative arguments of many kinds can reinforce each other. If social processes [DLP] and the evidence from technology that reactive systems and agents cannot be expressed by algorithms are taken into account, then the evidence for irreducibility is overwhelming.

G37: Show that irreducibility of mathematics to logic implies irreducibility of IMs to TMs.

Definability: irreducible things cannot be completely defined in terms of already existing things.

IMs are defined by an intensional class membership property with no explicit extension. The class of IMs is defined by an effectively testable property: the term "interaction machine" is an abbreviation of the term "machine with the interaction property". Intensional definition by an effective class membership criterion defines IMs as an incomplete, open class that reflects the incompleteness and openness of members of the class. Instead of defining a universal IM, we point to examples of IMs in the literature, like Milner processes, Lynch IO automata, and SIMs, that express interesting modes of interaction.

G38: Develop a framework for intensional definition of IMs and other irreducible entities.

Nonformalizability: sacrificing completeness for expressiveness requires new testing principles

Incompleteness requires the goal of proving correctness (reducing system behavior to formally specified behavior) to be replaced by weaker goals of testing, model checking, and result checking. Emphasis on formalizability/reducibility in early models of computing led to principles like "goto considered harmful" [Di] and the more sweeping "assignment considered harmful" of functional programming. Functional and gotoless programming enhance formalizability but restrict program structure and are harmful to ease of programming [Kn]. The restriction of intuitive computing to TMs is a more serious harmful consequence of formalization, since it reduces problem-solving power and expressiveness so that interactive systems like objects, personal computers, and networks cannot be adequately modeled. If silver bullets are interpreted as formal system specifications, then Fred Brooks' persuasive argument [Br1] that there is no silver bullet for specifying complex systems can actually be proved. Nonformal principles that abstract from

existing technology and support systematic design will be developed.

G39: Examine the impact of nonformalizability on the goals of formal methods and testing.

6. Design Space for Models of Interaction

Whereas the design space for research on algorithms is well structured, the design space for models of interaction is fragmented and not considered mainstream by algorithms researchers. We examine the design space of models of interaction to determine the impact of irreducibility in each area and the degree to which irreducibility can serve as a principle for establishing links among areas.

G40: Develop a roadmap of the design space for models of interaction.

Process Abstraction

Models of interaction help us understand distinctions among concurrent, distributed, and interactive abstractions. Concurrent abstractions deal with parallelism, distributed abstractions deal with logical and physical separation among components, while interactive abstractions deal with input-output behavior. Even the simplest interactive systems, like interactive identity machines that simply output what they input, have nonalgorithmic behavior [We1], while distributed and concurrent systems have interesting algorithmic behavior that is the subject of textbooks on parallel and distributed algorithms [Ly].

G41: Examine the relation of models of concurrency and distribution to models of interaction

Though Milner realized as early as 1975 [Mi75] that functions cannot express meanings of processes, work on process models in the 1980s focused on questions of concurrency and concurrent composition and it was not until the 1990s [Mi] that the focus shifted to interaction. The distinction between interactive and concurrent phenomena and the realization that interaction rather than concurrency is the source of non-functional behavior represents an important conceptual paradigm shift.

G42: Distinguish models of process concurrency from models of process interaction.

Both the Calculus of Communicating Systems (CCS) and the Pi calculus [Mi1] extend the algebraic elegance and parsimony of lambda reduction rules to interaction between a sender and a receiver, but identify processes by their properties rather than by a process name. The pi calculus was motivated conceptually by eliminating the distinction among values, operators, and processes in the spirit of Actors [Ag1], and practically by the need for mobile processes that dynamically reconfigure their interconnections.

G43: Contrast reduction rules and nonalgorithmic communication rules for CCS and the Pi calculus.

Interaction among processes is modeled by nonalgebraic unicasting protocols that express the anonymous binding of positive to negative ions of a chemical abstract machine, binding of complementary DNA sequences, and binding of sperm to eggs in reproduction. Anonymous communication among anonymous processes is useful in many empirical contexts, reflecting the fact that chemical and reproductive coupling is anonymous, and may be contrasted to the sending of messages to named receivers that characterizes the Actor model. Models of interaction provide a basis for distinguishing between the algebraic abstractions of process models and the open system abstractions of Actor models [Ag1].

G44: Contrast anonymous unicasting protocols with named receiver protocols of Actor models.

G45: Contrast algebraic process abstractions of [Mi] with open system abstractions of [Ag].

Models of interaction provide a framework for reinterpreting results on observational equivalence [Mi] and testing [DH, DDM] as extensions of algorithmic to interactive models of system observability. In [GW] we show that on-line observers can observe a greater range of behaviors than off-line observers. Off-line testers for algorithms were not designed to observe interactive system behavior just as humans are not designed to observe infrared images, and their failure to notice interactive behavior unwittingly strengthened the view that system behavior could be algorithmically described. Bisimulation (Bi) and the Brock-Ackerman anomaly (BA) are viewed in [GW] as modes of on-line observation and the proof that on-line testers are more expressive (more discriminating) than off-line corroborates the claim that systems have richer behavior than off-line testers. Systems have always had richer behavior than algorithms but until recently our mechanisms for observing systems were not rich enough to discover it.

G46: Develop methods of observing and testing the on-line interactive behavior, such as Bi and BA.

Bi has been studied for process models while BA has been studied for Actor models: comparison of testing models provides comparative insight into behavior of algebraic and open-system process models. Bi examines glass-box behavior by observation trees [DDM] that express structured operational semantics of state transitions. BA examines black-box observation and views as anomalous interactive distinctions unobservable by off-line observers. In [GW] we examine glass-box, black-box, and statistical modes of on-line observation, showing that on-line is more expressive than off-line observation in all three cases and proposing a systematic program for developing test methods for modes of interactive behavior.

G47: Classify modes of interactive system behavior by modes of observation and testing.

[DDM] provides a "structured operational semantics (SOS)" model of observability that expresses extensional observation structures by "decorated state-transition structures". Operational semantics is useful in defining restricted forms of interaction, but is inadequate as a general model of interactive behavior because observable behavior cannot in general be expressed by state-transition rules.

G48: Distinguish structural operational semantics from "denotational" interaction semantics.

Constraint and Logic Programming

Constraints can declaratively specify partial system behavior as in testing, interface, or query specifications, or imperatively constrain the space of all possible behaviors to a progressively smaller subset of desired behavior. Constraints make no assumptions about the inner structure of systems whose behavior is being constrained and are applicable to systems whose components are nonalgorithmic and noncompositional. [VS] documents the use of constraints in specifying databases, user interfaces, robots and a variety of other interactive applications. Specification by constraints can be viewed as a "sculpture" paradigm: System specification by progressively constraining possible behaviors mimics Michelangelo's creation of David by chipping a block of marble until the desired form emerged.

G49: Contrast constraint models of system behavior with state-transition models.

Constraint programming is the process of problem solving by successively imposing constraints until the solution emerges. Logic programming is a particular form of constraint-programming paradigm: unification is a mechanism for systematically adding incremental consistent constraints. The duality between constraint and state-transition problem solving is shown [Sa] by ask and tell versus read and write operations on memories. Viewing the store as a constraint rather than a valuation allows reading and writing to be replaced by order-independent ask and tell operations that can be applied concurrently. [SRP] develops semantic foundations for concurrent constraint programming in terms of process models [OH].

G50: Contrast reversible (backtracking) and committed-choice (reactive) constraint programming.

There is a fundamental distinction between constraints that are additive, order-independent, nondirectional, and declarative [VS] and nondeterminate committed-choice constraints of concurrent logic and constraint programming [Sh1, SRP]. The distinction between reversible (don't-know) and committed-choice (don't-care) nondeterminism captures the essence of declarative versus interactive control. Committed choice expresses object-oriented programming, interaction machines, and the irreversibility of time. We show in [We5] that commitment to a particular path in the choice tree (observation tree) is logically incomplete because solutions lying in the discarded part of the search space can never be recovered.

G51: Relate committed-choice logic and constraint programming to models of interaction.

The argument that completeness is incompatible with committed-choice (reactive) concurrent logic programming was used in 1992 at the closing of the Fifth-Generation Computing Project in Tokyo to show that the goal of "computing by logic" was unachievable even in principle [We5].

G52: Show that "computing by logic" is unachievable because logic is weaker than computing.

The semantics of incomplete information associated with constraints is very different for algorithmic and interactive models. Incomplete information that approximates algorithmically definable complete information can be expressed by Scott domains with fixed-point semantics [GS]. Fixed-point semantics expresses approximation to enumerably infinite behavior of closed systems: least-defined fixed points approach their limit point from the inside out while greatest-defined fixed points approach the limit point by outside-in constraints. In contrast, interactive models, including committed-choice domains of logic and

constraint programming [Sh1, SRP], have no fixed-point semantics. The distinction between incomplete interactive models with no fixed-point semantics and partial information models with fixed-point semantics expresses the distinction between undecidability and incompleteness.

G53: Explore the consequences of nonexistence of fixed points for partial information models.

Nondeterminism

Nondeterminism is an interactive abstraction that models incompleteness and uncertainty [WM]:
output nondeterminism: automata or humans may have multiple next actions
input nondeterminism: inputs of automata or humans cannot be predicted
underspecification: incomplete specification to permit flexibility of implementation
abstraction: ignore irrelevant detail by inner information hiding or outer context hiding
probability: nondeterministic alternatives have a probability distribution
commitment: committed-choice nondeterminism of concurrent logic/constraint programming
time: reduces nondeterminism by incremental commitment to an unpredictable and irreversible future

Output nondeterminism, which concisely models agents with multiple action choices, has been the most widely studied form of nondeterminism. Input nondeterminism is more central to the study of interaction because it directly models the uncertainty of interactive input. Underspecification of requirements and design is common in software engineering to delay binding of behavior and implementation. Abstraction expresses the essence of nondeterminism, ignoring properties irrelevant to a particular model in order to focus on properties judged relevant. Probability is a form of nondeterminism that assigns probability measures to nondeterministic alternatives and supports Bayes' theorem to determine posterior from prior probabilities when new information becomes available. Committed choice in logic programming, which irreversibly eliminates nondeterministic choices, is incompatible with completeness [We5]. Reversible (backtracking) nondeterminism is noninteractive and time-independent, while irreversible committed-choice nondeterminism expresses interaction and the irreversibility of time [Sh1, SRP, We5].

G54: Classify forms of nondeterminism by their conceptual and pragmatic roles in interactive models.

Time

Time is a defining abstraction for interactive behavior: functional behavior is independent of both the elapsed time required to compute a function and of the time at which the function is computed. Even complexity theory abstracts from elapsed time to model the number of computation steps, so that a hardware speedup does not affect the time complexity. Time can be modeled as an incrementally revealed abstraction like data abstraction that has properties like irreversibility [Pr2] that make it unique. Developing good conceptual and practical abstractions of time is an important topic of interactive research.

G55: Compare models and abstractions of time in the computer and physics literature.

Time is an elusive concept with properties like causality that can be modeled easily and other properties like duration, sharing, fairness, and real-time constraints that are hard to model. Lamport [La] shows that causal ordering, which specifies necessary ordering among events, is both easier to model and more significant than the actual ordering of events in a distributed computation. Defining events as concurrent if they are not causally related [Ga1] allows correctness of concurrent systems defined in terms of causality to be tractably analyzed. The design space for models of time includes models for causality [La] and temporal logic [MP] as well as the Chu Space models of Pratt [Pr1] that express the duality between inner states and external events, which is related to the duality between algorithms and interaction.

G56: Explore computational abstractions of time for causality, duration, irreversibility, duality, etc.

Real-time systems are inherently interactive and have nonfunctional dependability, availability, and quality-of-service requirements [St1]. Successful embedded real-time systems for mission-critical, industrial quality of service and embedded applications will be explored from the viewpoint of models of interaction to determine the role of simulation, testing, analytical methods, and other technology in realizing real-time performance guarantees. Since practice leads theory, bottom-up analysis of real-time systems

may turn out to be more relevant to building good systems than top-down principled design.

G57: Specify models and abstractions of time that are useful for real-time computing.

Specification and verification

[CW2] defines specification as precise behavior description and distinguishes between functional, timing, and performance behavior. It reviews case studies of system specification, verification, and model checking in the design and construction of reliable systems that provide valuable data for interactive specification and verification models. Though impossibility of complete formalization may appear to have a negative impact on specification and verification research, it in fact has a positive impact by providing a more realistic framework for analysis. Recognition that complete formalization is impossible allows the role of testing techniques like model and result checking to be better specified.

G58: Examine the role and application of model checking in case studies of interactive testing.

G59: Examine the impact of incompleteness on the goals and technology of formal methods.

Broy [Br3] has examined pre- and post-condition specifications to data abstractions like queues with the help of assumption/commitment (A/C) specifications, which have the following form for queues:

assumption: for all initial stream segments of queue and dequeue operations (q,d), #q >= #d

commitment: the output stream consists of the first n enqueued input stream elements, where n = #d

G60: Formalize the relation between pre and post conditions and A/C specifications.

A/C specifications generalize pre- and post-conditions to functions from input to output streams, reflecting that data abstractions specify functions from streams to streams though individual operations are not functions. Implementations of enqueue and dequeue operations illustrate the fact that code fragments corresponding to operations of data abstractions and objects are not functions from inputs to outputs.

Queues may be modeled by SIMs but not by algorithms. A/C specifications show that initial finite stream segments of SIMs determine functions and can be reduced to strings, but that noninitial substreams have context-dependent semantics. Research on effective techniques for gaining confidence in the adequacy of interactive programs includes work on testing, result checking, and constraints.

G61: Compare and contrast testing and result checking for gaining confidence in interactive programs.

On-line algorithms

On-line algorithms extend algorithm analysis from single algorithms to algorithmically specifiable patterns of algorithms, where complexity is measured by interaction cost rather than transformation cost. On-line algorithms for exploring graphs or maps have been studied extensively [PY]. The lower bound on interaction cost for this family of related on-line algorithms shows that complexity analysis of the number of interaction steps can replace complexity analysis of state-transition steps for certain classes of interaction problems. The class of interactive problems for which such analysis is appropriate, which includes algorithms for financial transactions [Ya] and other applications, should be precisely specified.

G62: Characterize the classes of problems for which on-line algorithms are useful.

An on-line process together with its environment has the closed-system property (csp) if its accessible environment e together with P form a closed system. Many on-line algorithms are specified as patterns of algorithm execution steps for process, environment pairs (P,e) with the csp. The lower-bound problem for finding a point on a line or an intersection in a Manhattan graph has the csp because the agent together with its data structure forms a closed system. (P,e) pairs with the csp can be viewed as TMs with a state-transition engine P and a tape e, where the cost of communication between P and e dominates the cost of state transitions. However, not all computations of systems with the csp are on-line algorithms since heuristics that make use of incrementally discovered information are interactive.

G63: Examine the closed-system property as a closure requirement for on-line algorithms.

Open systems

Openness can be precisely defined as interactiveness: systems are open iff they are interactive and

closed iff they are algorithmic. Open systems describe a larger class of problems than closed systems and let us constrain their behavior to define interesting modes of partial behavior. Concepts, models, and methods developed for interaction can serve as a basis for open system analysis. Interactive systems provide a concrete model for open systems in terms of which their properties can be concretely explored.

G64: Develop technical correspondences between properties of open systems and interactive systems.

Open systems can be closed by enlarging them to include all components with which they interact, while closed systems can become open by removing some of their parts. Opening a well-functioning closed system by removing a part, like removing a sparkplug from a well-functioning engine, can have disastrous consequences and cause regular (algorithmic) behavior to become unpredictable (nonalgorithmic). Closedness is a nonmonotonic property of systems in the sense that a closed system may have open subsystems. In fact, all closed systems are composed of open subsystems so that algorithm specifications are not "closed" under taking of subsystems. This lack of closure is similar to that of integers under division and of rationals under algebraic operations.

G65: Explore nonmonotonicity and nonenumerability properties of open and interactive systems.

References:

[Ag1] Gul Agha, *Actors: A Model of Concurrent Computation in Distributed Systems*, MIT Press, 1986.

[Ag+] Gul Agha et al., Abstraction and Modularity Mechanisms for Automatic Computing, in [AWY].

[AWY] Gul Agha, Peter Wegner, and Akinori Yonezawa, *Research Directions in Concurrent Object-Oriented Programming*, MIT Press, 1993.

[BK] Manuel Blum and Sampath Kannan, Designing Programs That Check Their Work, *JACM*, Jan. 1995.

[Br1] Fred Brooks, *The Mythical Man-Month: Essays on Software Engineering*, second edition, Addison-Wesley, 1995.

[Br2] Rodney Brooks, From Earwigs to Humans, MIT report, 1997.

[Br3} Manfred Broy, Draft book on the Focus Specification Language, Technical University, Munich, 1997.

[Br4] Manfred Broy, A Theory for Nondeterminism, Communication, and Concurrency, *Theoretical Computer Science* 45, 1986.

[CW1] Luca Cardelli and Peter Wegner, On Understanding Types, Data Abstraction, and Polymorphism, *Computing Surveys*, December 1995.

[CW2] Edmund Clarke and Jeanette Wing, Formal methods: State of the Art and Future Directions, *Computing Surveys,* December 1996.

[DDM] Pierpaolo Degano, Rocco DeNicola, and Ugo Montanari, Universal Axioms for Bisimulations, *Theoretical Computer Science*, June 1993.

[DH] Rocco DeNicola and Michael Hennessy, Testing Equivalences for Processes, *Theoretical Computer Science*, 34, 1984.

[Di] Edsger Dijkstra, Go To Considered Harmful, *CACM* 1968.

[DLP] Richard DeMillo, Richard Lipton, and Alan Perlis, Social Processes and Proofs of Theorems and Programs, *CACM*, May 1979.

[EKW] Oren Etzioni, Keith Golden, and Daniel Weld, Sound and Efficient Closed-World Reasoning for Planning, *Artificial Intelligence*, Vol 89 #1, January 1997.

[Ga2] Vijay Garg, *Principles of Distributed Systems*, Kluwer 1996.

[Gr] Ulf Grenander, *Lectures on Pattern Theory*, Vols 1,2,3, Springer Verlag, 1976.

[GS] Carl Gunter and Dana Scott, Semantic Domains, *Handbook of Theoretical Computer Sceince*, Volume B, Elsevier, 1990.

[GW] Dina Goldin and Peter Wegner, On the Expressive Power of Interactive Observers, submitted to ICALP 1998,, January 1998.

[He1] Carl Hewitt, Open Information Systems Semantics for Distributed Artificial Intelligence, *Artificial Intelligence*, 47, 1991.

[HL] Juris Hartmanis and Herbert Lin, *Computing the Future: A Broader Agenda for Computer Science*

and Engineering, National Academy Press, 1992.
[Kn] Donald Knuth, Structured Programming with Goto Statements, *Computing Surveys*, December 1994.
[La] Leslie Lamport, Time, Clocks, and Ordering of Events in a Distributed System, *CACM*, July 1978.
[LL] Leslie Lamport and Nancy Lynch, Distributed Computers: Models and Methods, *Handbook of Theoretical Computer Science*, Volume B, Ed. Jan Van Leeuwen, MIT Press/Elsevier 1990.
[LMWF] Nancy Lynch, Michael Merritt, William Weihl, and Alan Fekete, *Atomic Transactions*, Morgan Kaufman, 1994.
[Ma] David Marr, *Vision*, Freeman 1982.
[Mi1] Robin Milner, Elements of Interaction, *CACM*, January 1993.
[MP] Zohar Manna and Amir Pnueli, *The Temporal Logic of Reactive and Concurrent Systems*, Springer Verlag, 1992.
[Mu] David Mumford, Pattern Theory: A Unifying Perspective, *Proc 1sr European Congress of Mathematics*, Birkhauser Boston, 1994.
[Pa] Christos Papadimitriou, *Computational Complexity*, Addison Wesley, 1993.
[Pe2] Roger Penrose, *The Emperor's New Mind*, Oxford 1989.
[Pr1] Vaughan Pratt, Time and Information in Sequential and Concurrent Computation, boole.stanford.edu/chuguide/html#cks
[Pr2] Huw Price, *Time's Arrow*, Oxford 1996.
[PY] Christos Papadimitriou and Mihalis Yannakakis, Shortest Paths Without a Map, *Theoretical Computer Science* 84-1, July 1991.
[Re] Raymond Reiter, A Logic for Default Reasoning, *Artificial Intelligence*, Vol 13 #1, January 1980.
[Sa] Vijay Saraswat, *Concurrent Constraint Programming*, MIT Press, 1993.
[Sh1] Ehud Shapiro, The Family of Concurrent Programming Languages, *Computing Surveys*, Sept. 1989.
[SRP] Vijay Saraswat, Martin Rinard, and Prakash Panangaden, Semantic Foundations of Concurrent Constraint Programming, LICS 1982.
[St1] John Stankovic, Strategic Directions in Real-Time and Embedded Systems, in [WD].
[St4] Daniel Sturman, *Modular Specification of Interaction Policies in Distributed Computing,* PhD Thesis, CS Dept, University of Illinois.
[Tu] Alan Turing, Systems of Logic Based on Ordinals, *Proc. London Math Soc.,* 1939.
[VS] Pascal Van Hentenryck and Vijay Saraswat, Strategic Directions in Constraint Programming, in [WD].
[WD] Peter Wegner and Jon Doyle, Strategic Directions in Computing Research, Special Issue of *Computing Surveys*, December 1996.
[We1] Peter Wegner, Why Interaction is More Powerful than Algorithms, *CACM*, May 1997.
[We2] Peter Wegner, Interactive Foundations of Computing, *Theoretical Computer Science*, Feb. 1998.
[We3] Peter Wegner, Interactive Software Technology, *Handbook of Computer Science and Engineering*, CRC Press, 1996.
[We4] Peter Wegner, Concepts and Paradigms of Object-Oriented Programming, ACM *OOPS Messenger*, August 1990.
[We5] Peter Wegner, Tradeoffs Between Reasoning and Modeling, in [AWY].
[WM] Michal Walicki and Sigurd Meldal, Algebraic Approaches to Nondeterminism, *Computing Surveys*, March 1997.
[Ya] Ran El Yaniv, Competitive Solutions for On-Line Financial Problems, *Comp Surveys*, March 1998.