The open problem of open-world computing

Srinath Srinivasa

IIIT-Bangalore

Outline

Connotations of problem solving

- The term “Problem Solving” takes on different meanings when it is used colloquially and when it is sought to be formalized.
- Computational or *algorithmic* problem solving is the activity of problem solving using a computer (machine or mechanical methods).
- Some connotations of algorithmic problem solving:
  - Decision
  - Computation
  - Search

Connotations of problem solving

Decision

Decision is formally defined as a function of the form:

$$f : I \rightarrow \{yes, no\}$$

where I is the problem space.

- The "decision" problem solving assigns a value of either *yes* or *no* to every element in the problem space.
- Related connotation of *Recognition* that assigns a value *yes* to every valid element of the problem space.

Computation, Search

Computation

$$f : I \rightarrow O$$

Computation maps a problem from a given "problem space" I to a solution in the "solution space" O.

Search

$$f : S \times Q \rightarrow 2^S$$

Qiven a "query space Q, *Search* can be represented as a mapping from a search space S to one of its subsets $s \in S$.

Algorithmic problem solving

- Problem solving processes that can be written as closed-form functions (as in the previous examples) are also called *algorithmic* problem solving
- An *algorithm* represents a step-wise mechanical procedure that can compute the function f represented by the problem solving process
- For theoretical analyses, all algorithmic problem solving questions are reduced to the *decision* problem. For instance, the function:

$$f : I \rightarrow O$$

can be reduced to a decision problem as follows:

$$f : I \times O \rightarrow \{yes, no\}$$

Algorithms and sets

- The decision problem that is the cornerstone of algorithmic problem solving, can also be represented as the "subsethood" problem in set theory
- For instance, given a problem domain I, a decision problem of the form:

$$f : I \rightarrow \{yes, no\}$$

can be written as a set of the form:

$$I_f = \{x \mid x \in I, f(x) = yes\}$$

Algorithms and logic

- Axiomatic set theory from its genesis from Russel and Whitehead's Principia Mathematica, establishes close bindings between sets and First-order Predicate Logic statements
- A set of the form

$$X = \{x \mid P(x)\}$$

  is said to encapsulate elements of a given *type* whose properties are defined by the FoL predicate $P(x)$
- Given that algorithmic problem solving corresponds to the subsethood problem, we can now see that it corresponds to the problem of *logical entailment* of $P(I_f)$ that defines the properties of I_f starting from $P(I)$.

Turing Machines

- Turing Machines formalized the notion of *effective computation* or algorithmic problem solving

Turing Machine

A Turing Machine (TM) is specified as:

$$TM = (S, \Sigma, s_0, \delta, H)$$

where

- S is the state space of the computation
- Σ is the input alphabet
- $s_0 \in S$ is the starting state
- $\delta : S \times \Sigma \rightarrow S \times \Sigma \times \{L, R\}$ defines the TM dynamics
- $H \subset S$ defines the set of halting states

Turing Machines

- Given an alphabet Σ, the *Universe of Discourse* for a TM computation is defined as Σ^* where $*$ is the Kleene closure operator
- A TM computation starts with the TM head on the left-most end of an infinitely long tape. The tape comprises of a problem statement $w \in \Sigma^*$ occupying finitely many cells from the start of the tape
- The TM begins computation from state s_0 and at each computational step, makes a transition to one of the states $s \in S$, optionally writes back a character $c \in \Sigma$ onto the tape, and moves the head one cell to the left or right, as defined by δ
- The TM computation halts when it reaches one of the halting states $h \in H$.

Turing Machines

- Ironically Turing Machines were proposed to show that “effective computation” (or algorithmic problem solving) is not possible for all problems [Tur37]
- Alan Turing posited this in response to Hilbert’s 10th problem, famously called the *Entscheidungsproblem* (decision problem)

Church-Turing Thesis

The Church-Turing thesis states that if an effective computation (algorithmic problem solving) process terminates then there is an equivalent Turing Machine (or a λ-calculus or a recursive function) that exists for the process.

- The Church-Turing thesis is seen as the underpinning of what is theoretically computable by any computer and has near-universal acceptance

Uncomputability

- A major cornerstone of the theory of computation is the proof of existence of *uncomputable* problems
- Uncomputable problems are functions of the form $f : I \rightarrow \{yes, no\}$ for which a recursive function or a Turing Machine is shown to **not** exist

Uncomputability

- The proof of the existence of uncomputable problems derives insight from a very significant result in Set Theory due to Cantor

The proof comprises of two parts:

1. Show that the set of all computable problems are countably infinite
2. Show that the set of all possible functions of the form $f : I \rightarrow \{yes, no\}$ is uncountable when $|I| = \infty$

- This is shown by Cantor's theorem[1] that the power set of a set X is always bigger than X even when X is infinitely large
- Proof details out of the scope of this talk. (But I am always happy to discuss more!)

[1] Wikipedia page for Cantor's theorem:
http://en.wikipedia.org/wiki/Cantor%27s_theorem

Closed worlds

Classical axiomatic systems (and by implication, set theory and theory of computation) are called *constructivist*, *minimalist* or *closed-world* systems. The formalization of closed-worlds happen in different ways in each of these domains. We shall briefly survey them in the next few slides.

The Closed-world Assumption

Axiomatic systems based on first-order logic is based on the presumption that *what is not known to be true is false*.
Consider the following database table listing professors and their research interests:

Name	Research interest
R. Bera	Quantum computing
K. V. Dinesha	Software engineering
S. Rao	Distributed computing
D. Das	Wireless networks
S. Srinivasa	Databases

Now, a database query (say written in SQL) of the form: “Does R. Bera has a research interest in Intellectual Property Rights?” returns a response `false`. Strictly speaking, the answer is **unknown**, as it is neither *specified* nor *proven* to be false that R. Bera has a research interest in Intellectual Property Rights.

Atomic computations

A Turing Machine computation is said to be an *atomic* transition from the start state s_0 to the halt state as far as its *observable behaviour* is considered. It also means the following assumptions:

- The problem is specified in its entirety before the TM begins computation
- The problem statement does not change during the course of the computation

In database systems, such criteria is explicitly invoked on transactions in the form of the ACID (atomicity, consistency, isolation, durability) property.

Well-founded Sets

Axiomatic set theory (based on the most popular Zermelo-Fränkel axioms) are prevent circular subsethood conditions by the *Axiom of Foundation*.
Informally, the Axiom of Foundation states the following:

> *Every non-empty set contains an element that is disjoint from the set itself.*

The Axiom of Foundation also entails that no set is a member of itself.

Well-founded Sets

Because of the Axiom of Foundation, axiomatic sets require that any system of sets be constructed from a basic set of atomic “elements.” Every set theoretic construct that is possible in this world are only those that can be constructed from the elements.

Minimalist worlds

- A closed world is also called a *minimalist* world, since ignorance is assumed to be equivalent to falsity
- In other words: *In a closed world, everything is forbidden, unless explicitly allowed (specified in the axioms or entailed by them*
- Totalitarian metaphor and similar metaphors from legal systems, security policies, etc.

Computing in open worlds

- Most real-world problems however are posed in an *open-world* setting
- Not only are problems not completely specified at the beginning of a computation, problem statements may change even when the computation is underway
- Examples
  - Driving
  - Workflow
  - Logistics

Interactive Problem Solving

- Interactive problem solving is a process of mapping from a problem state to a solution state in the form of an interactive dialogue
- Each step of the interactive process is an atomic computation that maps from one intermediate state of the process to another
- Examples of interactive problem solving:
  - Control system
  - Reactive systems
  - Robotic navigation

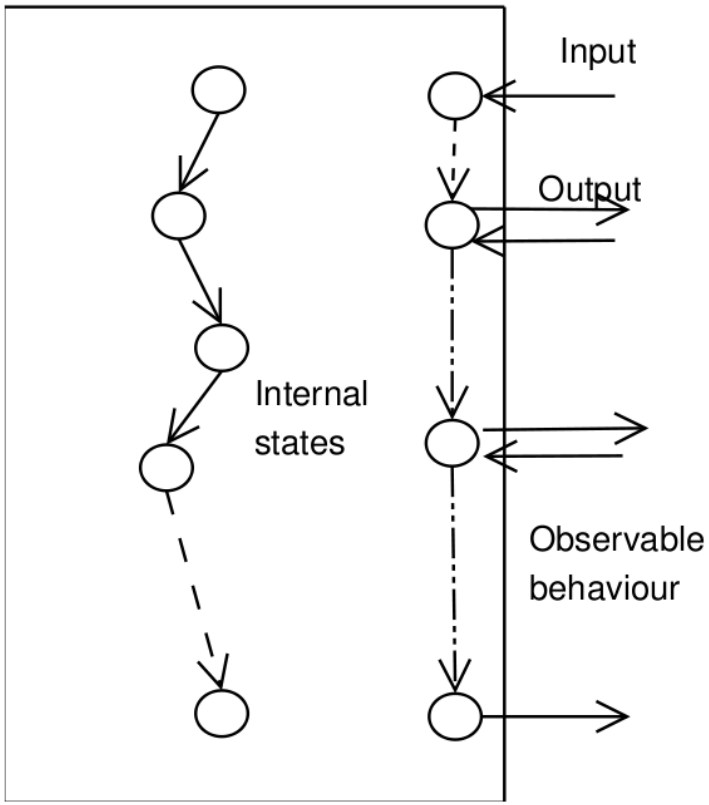
Interactive Problem Solving

- In an interactive dialogue, the computation at any interaction is determined not only by the inputs, but also the *current state* – or the *history of previous interactions*
- Hence an important property of interactive problem solving is: *persistence of state across computations*
- This is in contrast to algorithmic problem solving, where each computation starts afresh. For instance, a series of invocation of the function *sqrt*(9) will give the same observable behaviour; but a series of invocation of a function *front*(20) to a robot may not give the same observable behaviour. The starting state of the second invocation of the *front*() function is the ending state of the first invocation, and so on..

Interactive Problem Solving

- Intermediate inputs during an interactive process may depend on the intermediate outputs provided by the computation
- This makes it impossible to provide all possible inputs at the beginning of the computation
- Analogy with Heisenberg's uncertainty principle

Interactive Problem Solving



Interactive Problem Solving

Formalization

Interactive problem solving is defined by the following elements:

$$(S, I, O, \delta)$$

where

- S is a possibly infinite state space of the system in which the interaction happens
- I is the set defining the input alphabet from the environment
- O is the set defining the output alphabet from the machine
- $\delta : S \rightarrow (O \times I \rightarrow S)$ defines a computational transition from a given state, the previous intermediate output and the input from the environment to a target state.

Given a system as above, an interactive process is a *walk* in the state space of the form $s_1.s_2, \ldots s_n$ where for any $1 \leq i \leq n$, $s_i \in S$ and for any s_i, s_{i+1}, $\exists o \in O, i \in I$ such that $\delta(s_i) = f(\cdot, \cdot)$ where $f(o, i) = s_{i+1}$.

Open-world assumptions

- The challenge with interactive problem solving is that it is impossible to determine the trajectory of computation at the start of the process
- At an intermediate state, the environment can provide *any* possible input, unless we know explicitly that a specific input cannot be provided at that stage
- This encapsulates the *open-world assumption*:

A predicate P is deemed to be false iff it is explicitly stated as false or can be entailed as false from the known set of axioms

Open-world computing

- Axiomatic systems with the open-world assumption are called "maximalist" systems
- Colloquially: *Everything is allowed unless explicitly forbidden*
- Analogies from democracies, legal systems, workflow policies, etc.
- We survey the open-world assumption in different domains (logic, sets and computing) in the next few slides

Reasoning in open-worlds

- Logical deduction in closed worlds are *monotonic* in nature
- Establishing a conclusion will not refute any previously established conclusions
- On the other hand, relaxing the closed-world assumption requires reasoning processes to be *non-monotonic*. New observations that enter the system after the reasoning process has begun, may falsify previous conclusions and their entailments

Non-monotonic reasoning

Default Logics

One way to handle open worlds is to relax the tautological properties of axioms with the "default" property.

Hence, if a classical logic axiom said: $isBird(P) \Rightarrow canFly(P)$ (*All birds fly*), it can be relaxed with default logics to say: *By default, all birds fly (unless observed otherwise)*

A default theory is a pair (D, W), where W is the classical set of axioms forming the *background theory* and D is a set of *default rules*. A default rule is of the form: $Prerequisite : Justification_1, \ldots Justification_n \Rightarrow Conclusion$

For example, the default rule:

$$isBird(x) : \neg(\neg canFly(x)) \Rightarrow canFly(x)$$

is read as: *If x is a bird and there is no axiom (in W) or a prior conclusion that x cannot fly, then we conclude that x can fly*.

Non-monotonic reasoning

Abductive reasoning

Abductive reasoning is the process of building a hypothesis from a set of observations, in contrast to reaching a conclusion from a set of axioms.

Open-world assumption and non-monotonicity are integral aspects of abductive reasoning.

Given a *base theory* T and a set of observations O, abductive reasoning generates a set of hypotheses $\mathcal{E}$ such that for any $E \in \mathcal{E}$:

- E and T entail O
- E is consistent with T

Usually a "best theory" is chosen from the set of generated theories $\mathcal{E}$ based on meta-heuristics like: maximum entropy, minimum description length, universality, etc.

Non-wellfounded Sets

Well-founded sets do not allow for recursive constructs in set-theoretic statements that are essential for hypothesis construction.

For instance, when parsing a natural language text, consider that we represent each sentence by a set representing all the entities that it refers to. Now consider a sentence of the form:

> Sentence P: This sentence P is a statement written in the English language.

The set-theoretic representation of P is given as:

$$P = \{P, English\}$$

While such a construct is perfectly meaningful, it is forbidden by axiomatic set theory due to the Axiom of Foundation!

Non-wellfounded Sets

Hypersets

A *Hyperset* or a non-wellfounded set is a set that relaxes the foundation axiom (FA) and replaces it with the anti-foundation axiom (AFA). AFA allows for circular and infinite membership chains as long as they don't lead to a paradox [Acz88].

Thus:

- $$X = \{x \mid x \notin x\}$$

  (Russel's paradox) is an invalid Hyperset, while

- $$X = \{x \mid x \in x\}$$

  is a valid Hyperset

Circular membership is crucial for formalizing model building and for infinite computational processes like reactive systems.

Reactive Systems

- Reactive systems are systems that are meant to *maintain an interaction* with the external environment [MP92]
- Unlike classical connotations of problem solving, reactive systems don't have an end state and are not transformations
- Examples: Control systems, embedded controllers, listener/responders, etc.
- The behavior of reactive systems cannot be reduced to a single TM computation. Instead, they are modeled in the form of *state transition diagrams*

Reactive Systems

Labeled transition system

A labeled transition system (LTS) is defined as:

$$LTS = (S, \Sigma, \rightarrow)$$

where S is a (possibly infinite) state space, Σ is a set of labels (usually representing I/O details) and $\rightarrow \subseteq S \times \Sigma \times S$ denotes transitions between states.

An LTS is distinguished from a finite state machine in the following ways:

1. The state space (and the set of transitions) of an LTS need not be finite, or even countable
2. The LTS does not start from the same start state for every computation

A reactive computation constitutes a (possibly infinite) walk in the state space of the LTS. Conventional reasoning is not sufficient to reason about and compare reactive systems. Dual mathematical models in the form of *co-algebra* and *bisimulation* have been developed for this purpose.

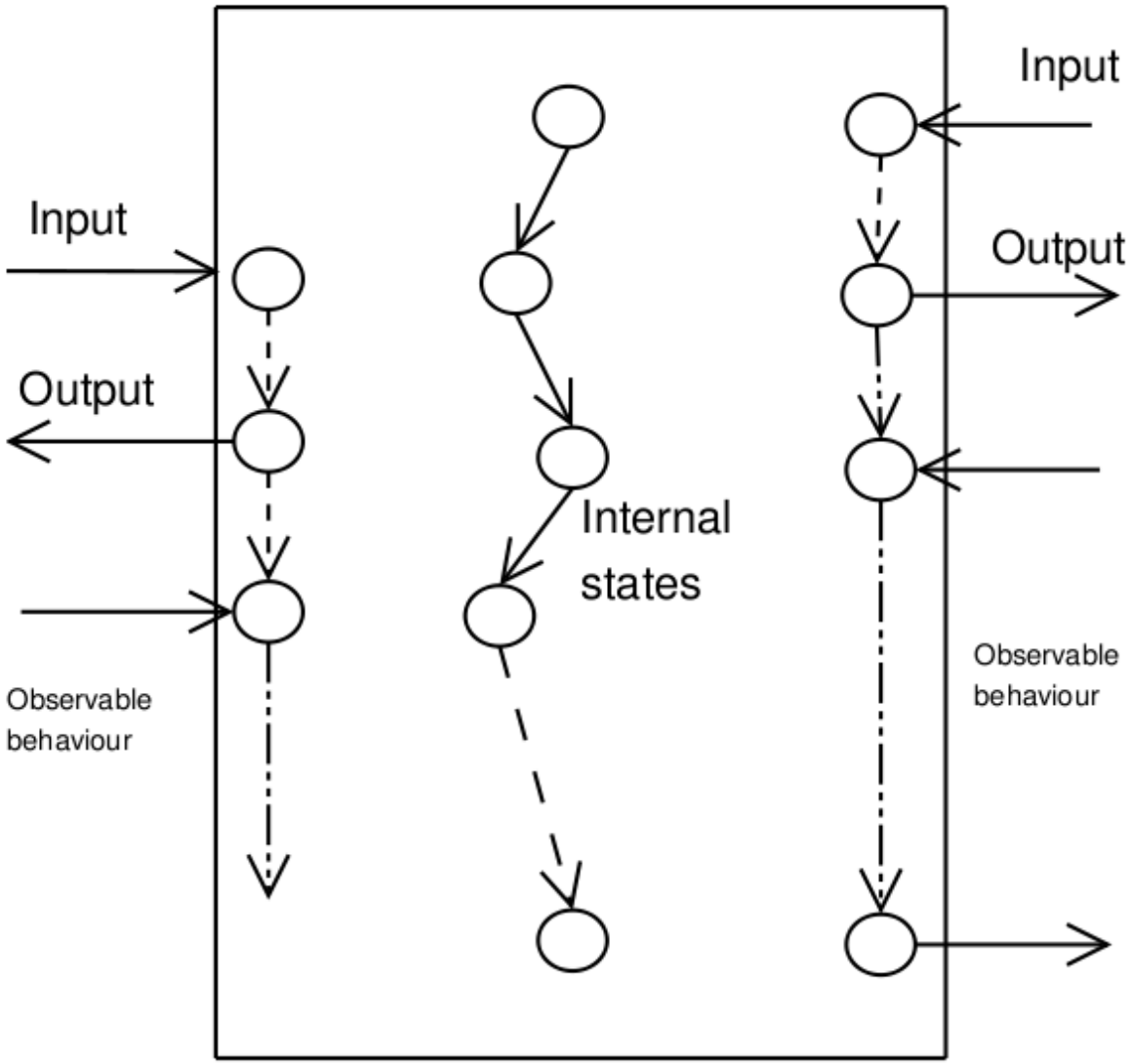
Questions for open-world computing

- Establishing liveness: When do we say that an interactive process (especially an infinte process) has "succeeded"?
- Providing guarantees: Can we say that a given interactive system will "work" at all? Especially since anything may happen at any time during the computation?
- Expressiveness: Is interactive computing more expressive than Turing Machines? For instance, can we solve a problem that is provably uncomputable, using interactive computing?

Multi-stream Interaction

- The story of open-world computing becomes even more intriguing when concurrent interactive computations are considered
- A well-known result in the theory of computation is that a Turing Machine having multiple tapes (MTM) is no more expressive than a Turing Machine operating over a single tape
- In 1997, Wegner and Goldin [WG97] contended that multi-stream interactive computations are strictly more expressive than single-stream interactive computations, as long as the open-world assumption is considered.

Multi-stream Interaction



Multi-stream Interaction

- A multi-stream interactive machine (MIM) interacts over multiple streams at the same time
- Since these processes operate in open worlds, they are not atomic transitions nor isolated from one another
- Observable behaviour on any one stream is determined by:
  - Inputs
  - History of interaction
  - Interactions happening on other channels
- While history sensitive responses of single-stream interaction machines are called *hidden variable* systems, MIMs are termed *hidden-adversary* systems.

Multi-stream Interaction

- Although Wegner and Goldin did not provide a proof for their conjecture, a number of examples help suspect that the assertion is true:
- Examples:
  - Interleaved and non-serializable transactions
  - Playing chess against two grandmasters
  - Passing Turing's test with MIMs
- While TMs are considered the mathematical foundation of algorithms, MIMs are considered to be the mathematical foundation (if and when a formalism is found) for database-backed information systems [GST00]

The Road Ahead

- Interference schema
- MIMs and the evolution of cooperation
- Bridging open-world computing and multi-agent systems
- Reasoning about MIMs


Peter Aczel.
Non-well-founded sets.
CSLI Lecture Notes, 14, 1988.


Dina Goldin, Srinath Srinivasa, and Bernhard Thalheim.
Information systems = databases + interaction.
In *Proceedings of the International Conference on Conceptual Modeling (ER 2000)*, 2000.


Zohar Manna and Amir Pnueli.
The Temporal Logic of Reactive and Concurrent Systems.
Springer-Verlag, 1992.


Alan M. Turing.
On computable numbers, with an application to the entscheidungsproblem.
Proceedings of the London Mathematical Society, 2(42):230–265, 1937.


Peter Wegner and Dina Goldin.

Why interaction is more powerful than algorithms?

Communications of the ACM, May 1997.