6. Models of interactive computing

1. Algorithms vs. interaction
2. Basic models of interaction
3. Persistent TMs and equivalent models
4. Expressiveness of PTMs

Inquiry

- What’s a paradigm shift?
- Is interaction more powerful than algorithms?
- Are there models of interaction that can solve problems that TMs can’t?
- Does the Chomsky hierarchy require extension?


Topic objective

Describe models of interactive computation and expressiveness results for them

Subtopic objectives

6.1a Distinguish algorithmic from interactive computation*

6.1b Explain or write a coinductive definition

6.2 Define an instance of a model of interaction

6.3 Describe a model of sequential interactive computation*

6.4a Describe expressiveness of interactive models

6.4b Relate sequential interaction to synchrony
1. Algorithms vs. interaction

- What’s concurrency?
- Is interaction the same as repeated input/output?
- Is communication part of computation?

Subtopic objectives

6.1a Distinguish algorithmic from interactive computation*
6.1b Explain or write a coinductive definition
Flowcharts show algorithms or interaction

Algorithm

Interactive process

Use cases show the user’s role in interaction

- Algorithms make no assumptions about users
- The *Unified Modeling Language* was widely adopted in the 1990s to account for interactive systems
A gap between theory and practice

Theory: algorithms only
Practice: interaction and algorithms

Definition

Algorithmic computation:
The effective transformation of a finite, pre-specified input, to a finite output, in a finite number of steps. (D. Knuth)

- Algorithms compute functions
- A system that executes an algorithm is closed
Alternative definition

- Algorithmic computation is, equivalently:
  - Halting Turing-machine computation
  - Halting computation by random-access machine, e.g., in $S$ language
  - Evaluation of $\mu$-recursive functions
- Derived from the Church-Turing thesis
- Note that these definitions exclude all interleaving of input and output during algorithm execution

Do TMs model all computation?

- “A Turing machine can do everything that a real computer can do” (Sipser, 1997, p. 125)
- TMs and decidability address the question “tantamount to the question of what computers can do” (Hopcroft, Motwani, Ullman, 2007, p. 315)
- TMs are “a veritable formalism for computers”; an “accurate model for what any computing device is capable of doing” (idem.)
A paradigm shift to interaction?

Features of computing today:

- Computation that is an ongoing *service*, not assumed to terminate
- Dynamic, alternating input and output that occur *during* computation
- *Persistence of state* (memory) between interaction steps

Intelligence and interaction

- *Thesis*: Intelligence is developed chiefly by and for solving interactive problems
- *Learning* consists of adaptation in a complex, partially observable, stochastic environment
- Intelligent agents conduct searches for a satisfactory *policy*, consisting of responses to an environment state, based on *percepts* that point to *rewards*
Kinds of concurrency

- *Multitasking and threads* may occur on a single processor
- *Sequential interaction* repeats input/output
- *Parallel and distributed computation* involves coordination among processors
- *Multi-stream interaction* (topic 7) brings together more than two computing entities and may entail *indirect interaction*

Communication

- *One-way communication* is the sending of strings, over a finite alphabet of symbols, from one computing entity to another
- *Two-way communication* is the concurrent activity of two entities engaged in one-way communication with each other
- Either entity *may* wait for input before emitting output
- Either entity *may* communicate exclusively with the other.
Interaction and synchrony

- Direct interaction is two-way communication in which some outputs of each entity may causally affect the entity's later inputs from the other.

- Entity A interacts synchronously with environment E if:
  - A interacts with E and
  - A and E wait for input before outputting.

- Asynchronous interaction occurs in the absence of synchrony as defined here.

Sequential interaction

Sequential interactive computation: Synchronous interaction involving two participants, at least one of which is a computing agent or entity.

- Characterized by a single interaction stream of input alternating with output.
- If one participant is an agent, the other is its environment.
- Interaction may involve changes of state.
**Autonomous agents and models of sequential interaction**

- Unified Modeling Language (UML) models sequential-interactive systems not supported by algorithm-based notations like flowcharts, module hierarchies, pseudocode
- *Autonomous agents* may initiate actions; may or may not synchronize with environment
- They may initiate *observation* actions
- To model autonomous agents, standard UML must be extended

**Contributions to the theory of interactive computing**

- *c*-machines (Turing), finite transducers (Moore)
- Cybernetics: models of feedback systems (Wiener)
- Information/communication theory (Shannon)
- Concurrency with message passing: CSP (Hoare), CCS (Milner), \( \pi \) calculus (Milner)
- I/O Automata (Lynch), Abstract State Machines (Gurevich), Site Machines (van Leeuwen, Wiedermann)
- Interaction Machines and Persistent Turing Machines (Wegner, Goldin)
Streams and coinduction

- Interactive programs continue outputting as long as they receive input
- Their behaviors are sets of infinitely long *streams* of input/output
- Infinite I/O can be described mathematically

Streams

- A reactive system may receive an infinite stream of inputs and may emit the corresponding infinite stream of outputs
- *Sets of streams* express interactive behavior
- *Coinduction* defines sets of infinite objects as induction defines sets of finite ones
Inductively defined sets

- (i) 0 is a natural number;
- (ii) Every \( n \in \mathbb{N} \) has a unique successor, \( n' \);
- (iii) i and ii are the only ways to obtain a natural number (Peano)

- Set of expressions using numerals, +, and ( ):
  \[ expr \rightarrow \text{numeral} \mid \text{numeral} + \text{expression} \mid (\ expr\ ) \]

- The definition of such a set may use itself
- But such a set does not contain itself

Inductive definitions of strings and languages

- String: a finite sequence of symbols
- \( \Sigma^* = \{ \lambda \} \cup \{ ax \mid a \in \Sigma, x \in \Sigma^* \} \)
  (note base case \( \lambda \))
- “\( L \) is a language over alphabet \( \Sigma^* \)” means that \( L \subseteq \Sigma^* \)
Coinductive definitions of languages

- **Stream**: an infinite sequence of symbols
- A *stream language* is a set of streams
- Example, defined coinductively:
  \[ \Sigma^\infty = \{ ax \mid a \in \Sigma, x \in \Sigma^\infty \} \]
  (note lack of base case)

Well-founded sets

- (B. Russell) Let the village hair stylist be the person who cuts the hair of everyone who doesn’t cut own hair
- Question: Who cuts the hair-stylist’s hair?
- Similar Questions: Is there a set of all sets? If so, does it contain itself? Is there a set of all sets that don't contain themselves?
- In classical (well founded) set theory, it is meaningless to say a set belongs or doesn't belong to itself (Foundation Axiom)
Another diagonal proof

- **Theorem:** The notion “set of all sets” is contradictory: i.e., meaningless
- **Proof:** Consider $\Omega' = \{ A : A \not\in A \}$ (sets not members of themselves)
- $(\Omega' \in \Omega') \Rightarrow (\Omega' \not\in \Omega')$,
  $(\Omega \not\in \Omega) \Rightarrow (\Omega \in \Omega)$, contradictions
- Note role of “Spoiler” instance $\Omega'$
- We say that $\Omega, \Omega'$ are *not well founded* (NWF)

Streams and non-well-founded sets

- A non-well-founded set may contain itself, directly or indirectly
- **Example:** $A = \{ B, C \}; B = \{ A, D \}$
- Every stream of characters is a character, followed by a stream of characters
- Streams contain streams, which contain streams, which contain streams, ...
- Ex.: The set of all bit streams $\{0,1\}^\infty$ consists of any infinite sequence that consists of 0 or 1 followed by a stream
Streams and coinduction

- **Induction** defines countable sets of finite objects: $\Sigma^* = \{\lambda\} \cup \{ax \mid a \in \Sigma, x \in \Sigma^*\}$

- **Coinduction**
  - is a dual of induction (recursion), lacking a base case
  - is used to define sets of infinite objects

- The set of streams over an alphabet, expressing ongoing processes such as interaction:
  $\Sigma^\infty = \{ax \mid a \in \Sigma, x \in \Sigma^\infty\}$

Least and greatest fixed points

- A *fixed point* $x$ of function $f$ is a value $x$ (which may be a set), for which $f(x) = x$

- Whereas induction is characterized by *minimality* conditions (*least* fixed points), coinduction has a *maximality* condition (*greatest* fixed point)

- **Minimality example**: $\Sigma^+$ is the *smallest* set that fits the spec $\{ax \mid a \in \Sigma, x \in \Sigma^*\}$

- **Maximality example**: $\Sigma^\infty$ is the *largest* set that fits the spec $\{ax \mid a \in \Sigma, x \in \Sigma^\infty\}$
2. Basic models of interaction

- What is a *reactive system* or a *control system*?
- Is *transduction* algorithmic or interactive?
- What is a *stream*?

Subtopic objective

6.2 Define an instance of a model of interaction
Models

- *Finite transducers* extend DFA model by specifying output at each state or transition
- *Kripke structures* model reactive systems for verification
- Turing’s *choice-machine* model allowed external choice of transition steps during computation
- Semantics of transducers could be dynamic or buffered I/O; dynamic = interactive

Interactive finite transducers

- See *Mealy machine* (topic 2)
  \[ M = \langle Q, \Sigma, \Gamma, \delta, \text{out}, q_0 \rangle \]
- *Semantics* of interactive finite transducers differ from algorithmic ones
  - Input and output are interleaved
  - Interaction is non-halting
- *Stream language* of a MM is a set of streams \( SL \subseteq (\Sigma \times \Gamma)^\infty \)
Example: soda machine

- Inputs (left side of labels): \{Q, $1\}
- Outputs (right side of labels):
  \{“25”, “50”, “75”, [soda]\}
- Observable behavior is the machine’s set of responses to inputs of currency

Example: digital clock

- Inputs: \{ tick \}
- Outputs: \{“12:00:00”, “12:00:01”…\}
- One state per time value
- With alarm, \((24 \times 60 \times 60)^2\) states

Other examples:

- Calculator
- Microprocessor
- Memory chip
- Any digital control device
Stream I/O

- Transducers such as Mealy machines model interactive devices: ICs, controllers, etc.
- Let $\Sigma$ and $\Gamma$ be input and output alphabets
- A *Mealy machine* has behavior described by the stream set $L \subseteq (\Sigma \times \Gamma)^\infty$ where
  $$(\Sigma \times \Gamma)^\infty = \{ ps \mid p \in \Sigma \times \Gamma, s \in (\Sigma \times \Gamma)^\infty \}$$
- $(\Sigma \times \Gamma)^\infty$ is the set of *streams over* $\Sigma \times \Gamma$
- An *I/O stream* is an I/O pair followed by a stream

Stream languages of finite transducers

- Stream language of *soda machine* is all streams that contain ($1, \text{soda}$) or ((Q, 25), (Q,50), (Q, 75), (Q, soda))
- “(Q, 25)” means (input quarter, output “.25”)
- Stream language is:
  $$((\$1, \text{soda}) \mid ((Q, 25), (Q,50), (Q, 75), (Q, \text{soda})))^\infty$$
Related ideas

- Transducers were once called “sequential machines” and were part of Curriculum 68, the first ACM CS curriculum
- Not part of standard theory texts today
- Related areas:
  - Markov decision processes,
  - model checking of reactive systems,
  - temporal logic, e.g, CTL, whose operators apply to streams

Kripke structures

- Unlabeled transition system, used to diagram and reason about reactive systems
- To a Kripke structure corresponds an infinite computation tree reflecting all possible paths through the system
Computation Tree Logic

- **State** formulas express assertions about a state (see labels on states of statechart below)
- **Path** formulas express assertions about a set of paths in a computation tree

![Statechart diagram]

**Key**
- `start` = start-button pressed
- `heat` = heating on
- `shut` = door shut

Application: model checking

- Verification of reactive systems may be by use of logic in **model checking**
- This process uses formulas in the CTL language about properties of the infinite computation trees observable on a reactive system
- Attributes verifiable: **liveness** (some desired event will eventually occur); **safety** (some unwanted event will never occur)
Markov models

- A *Markov state machine* or chain is a system with a finite number of observable states, and with probabilistic transitions between states.
- Defined by initial state $s_0$, transition model $T(s, a, s')$, and reward function $R(s)$.
- *Example:* weather at any location.

Example: Weather

- Let states be \{sunny, cloudy, rainy\}.
- Let transition probabilities be in labels.
- First-order Markov model:
A computer’s components interact

- The composition of two sequential-interactive systems may be a closed, algorithmic system
- A processor does not necessarily execute an algorithm in the fetch-execute cycle:
  - Interleaved I/O may be involved
  - A loop may not terminate
- A processor, its registers, and RAM interact with each other

3. Persistent TMs and equivalent models

- Does a TM store information between executions?
- What is the duration of a RAM’s memory?
6. Models of interactive computing

Subtopic objective

6.3 Describe a basic model of sequential interactive computation*

Definition of PTM model

- A minimal extension of TMs expressing *sequential interactive behavior* (Goldin, et al)
- A *PTM* is a 3-tape TM with $x_i$ $y_i$
  - Tapes for input, output, work
  - I/O as dynamically generated *streams of interleaved inputs and outputs*
  - TM executions (*macrosteps*) iterated
  - A persistent work tape, called a *memory*, which is preserved between macrosteps
6. Models of interactive computing

**PTM macrosteps**

- At each macrostep, PTM $A$ computes a TM-computable function $f_A$ from (input, worktape) to (output, worktape)
- Each TM computation takes an input, gives an output, and updates the work tape

Example: answering machine

- An *answering machine* $A$ is a PTM whose work tape contains a sequence of recorded messages
- Operations are *record message*; *playback*; *erase*
- Its TM-computable function $f_A$ is:
  - $f_A(\text{record } Y, X) = (\text{ok}, XY)$
  - $f_A(\text{playback}, X) = (X, X)$
  - $f_A(\text{erase}, X) = (\text{done}, \lambda)$
- For input stream that begins $(\text{record } m_1, \text{ erase, record } m_2, \text{ playback})$, $A$ generates $(\text{ok, done, ok, } m_2)$, where $m_1, m_2$ are messages
Stream behavior of PTMs

- The *persistent stream language* (PSL) of a PTM: the set of streams \( L \subseteq (\Sigma^* \times \Sigma^*)^\infty \) observable on it
- The set of all I/O streams over alphabet \( \Sigma \):
  \[ (\Sigma^* \times \Sigma^*)^\infty = \{ (a, x) \mid a \in (\Sigma^* \times \Sigma^*), x \in (\Sigma^* \times \Sigma^*)^\infty \} \]
- Persistent stream language (PSL) of a PTM \( M \): the set of interaction streams \((i_1, o_1), (i_2, o_3), \ldots\) where \(i_n, o_n \in \Sigma^*\), in which for every \(k\), there are memories \(w, w'\) such that \(f_M(w, i_k) = (o_k, w')\)
- The set of persistent stream languages is \(PSL\)

Stream languages and equivalence

- Two PTMs \(M_1\) and \(M_2\), are *stream (observationally) equivalent* iff \(PSL(M_1) = PSL(M_2)\)
- Two memories of \(M\), \(w_1\) and \(w_2\) are equivalent iff sub-PTMs with \(w_1\) and \(w_2\) as starting memories have the same PSL
- *Isomorphism*: the states have a structure-preserving bijection; design and behavior are the same
- *Bisimulation*: a deeper form of equivalence than observational, but short of isomorphism
Amnesic PTMs

- **Defn.**: PTMs that do not use their work tapes
- **Example**: squaring machine \((out_i = in_i^2)\)
- APTMs extend TMs with stream-based semantics, but do not make use of their memory, i.e., are equivalent to TMs iterating
- Unlike PTMs, they lack persistence.
- **ASL**: The set of *amnesic* stream languages, i.e., the languages of amnesic PTMs
- In an ASL, unlike PSL, order matters, because no macrostep affects later ones

Interactive extension to RAM

- Extends **S** language by use of variables \(W_1 \ldots W_n\)
- Unlike \(X, Y, Z\) variables, \(W\) variables retain their values after execution of a RAM computation
- Hence input and output of the RAM are interleaved, and \(W\) variables express persistent state
Mealy machine extension

- Consider a Mealy machine with an \textit{infinite} token set; \textit{infinite} set of states and transitions
- This automaton precisely simulates a PTM if we establish mappings
  - between the infinite set of states and the PTM’s set of possible work tape contents
  - between input, output tokens and the infinite set of I/O strings

Interactive Turing machines

- Turing defined an interactive version of the TM, called \textit{c-machines} (choice machines)
- A c-machine may accept input at any time
- PTMs are a convenient extension to \textit{c}-machines, formalizing input with an input tape and separating interactive computation steps
Interactive extension to $\mu$-recursion

Given a $\mu$-recursive function $\phi : \mathbb{N} \to \mathbb{N}$, construct function $f_\phi : \mathbb{N} \times \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ where

- The first parameter passed to $f_\phi$ is the parameter passed to $\phi$; second parameter is 0, or obtained from a previous evaluation of $f_\phi$
- The first element of the pair returned by $f_\phi$ is its first parameter on the next evaluation of $f_\phi$
- The second element returned is later passed to next evaluation as parameter 2

Proof challenge

- Can you prove that the following models are equivalent to the PTM in computing power?
  - Minimal PTMs
  - Interactive RAMs
  - Mealy-machine extension
  - Interactive extension to $\mu$-recursion
- That is, can you prove that each is more expressive than (RAM, TM, $\mu$-recursion)?
4. Expressiveness of PTMs

- Are there limits to the expressiveness of the TM or the RAM?
- Can a TM emulate a PC?
- What can be proven about the expressiveness of interactive computing?

Subtopic objectives

6.4a Describe expressiveness of interactive models*
6.4b Relate sequential interaction to synchrony
Goldin et al results
- We show that the PTM model is more expressive than the amnesic PTM, by the $\mathcal{ASL} \subseteq \mathcal{PSL}$ theorem (Goldin et al, 2004)
- But an APTM computation is equivalent to a TM computation, iterated
- Hence we say that the 2004 proof establishes that PTMs are more expressive than TMs

K-prefix languages for PTMs
- The $k$-prefix language of PTM $M$, $L_k(M)$, is the set of initial sequences of length $\leq k$ for streams in $\mathcal{PSL}(M)$.
- Corresponding notion of equivalence: we write $(M_1 =_k M_2)$ iff $L_k(M_1) = L_k(M_2)$
- $\mathcal{ASL} = \{\mathcal{PSL}(M): M \text{ is an amnesic PTM}\}$
- $\mathcal{PSL} = \{\mathcal{PSL}(M): M \text{ is a PTM}\}$
Theorem: $\text{ASL} \subset \text{PSL}$

1. Note that for any amnesic PTM $M$, if the 1-prefix language $L_1(M) = \{(x,y) \mid y = f_M(x)\}$, then $L_2(M) = L_1(M)^2$, $L_k(M) = L_1(M)^k$, and $\text{PSL}(M) = L_1(M)^*$

2. For APTMs, $(M_1 =_k M_2)$ iff $(M_1 =_1 M_2)$ iff $(M_1 =_{\text{ASL}} M_2)$

3. So for APTMs, the equivalences $(=_1)$, $(=_k)$, $(=_{\text{ASL}})$, are all the same

$\begin{array}{|c|}
\hline
_1 & =_{\text{ASL}} \\
\hline
\end{array}$

$\text{ASL} \subset \text{PSL}$, cont’d

4. Since output of a PTM at step $i$ is not dependent on input only, but also on state, PTMs in general have an equivalence hierarchy of $k$-prefix languages:

$\begin{array}{|c|}
\hline
_1 & \supseteq & _2 & \supseteq & \cdots & \supseteq & _{\text{PSL}} \\
\hline
\end{array}$

5. Hence there are PSLs that are not ASLs

$\begin{array}{|c|}
\hline
\text{ASL} & \subset & \text{PSL} \\
\hline
\end{array}$

(Goldin, Smolka et al, 2004)
Asynchronous PTMs

- **Definition**: asynchronous PTMs are PTMs that may not wait for partner to finish
- **Proposition**: PTMs with asynchrony are more expressive than (synchronous) PTMs
- **Rough proof sketch**:
  - Uses nondeterminism of asynchronous interaction
  - E.g., tokens may be lost in asynchronous interaction

Limitations of PTMs

- PTMs require *synchronized* I/O; that is, each participant in an interactive computation must wait its turn
- A PTM can interact only with environments that resolve their outputs to synchronized tokens; hence only environments of *non-autonomous* agents
- PTMs imply a *message-passing* model of interaction
6. Models of interactive computing

References on interaction


Peter Wegner. Why interaction is more powerful than algorithms. CACM 40 (5), 1997.

References on coinduction

Barwise-Moss. Vicious Circles, 1996
Finsler, 1920s (NWF sets)
Peano, 1889 (induction)