1. Algorithmic and serial computation

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

- Algorithms *compute functions*
- A system that executes an algorithm is *closed*
- Algorithms are equivalent to Turing-machine computation
Alternative definition

• Algorithmic computation is, equivalently:
  – Turing machine computation
  – Computation by random-access machine, e.g., in $\lambda$ language
  – Computation of $\mu$-recursive functions
• This definition is derived from the Church-Turing thesis
• Note that this definition excludes all interleaving of input, output, and processing steps

Serial computing

• The 50-year-old Von Neumann architecture defines most computing today:
  - one processor
  - data is in memory
  - program is in memory
  - enables general-purpose computing
• The microprocessor carries out a fetch/execute cycle, retrieving and executing machine instructions one at a time
• Multi-core machines break the one-processor assumption
Modeling concurrency

- **Interactive computation** alternates input with output
- **Multi-stream interaction** may bring together more than two computing entities and may entail *indirect interaction*
- **Parallel computation** may be algorithmic but involves interaction and sharing of memory among processors
- **Distributed computing** involves communication at a distance
- **Proposition**: Turing machines model none of these adequately

2. Sequential interaction

- Feature of computing today: Computation as an ongoing *service*, not assumed to terminate
- Dynamic input and output *during* computation
- **Persistence of state** (memory) between interaction steps

![Diagram of sequential interaction](attachment:image)
Extending automata models

- *Finite transducers* extend DFA model by specifying output at each state or transition
- Turing’s *choice-machine* model allowed external choice of transition steps during computation
- *Idea*: extend PDA, TM models to enable output at any transition: *pushdown transducer, interactive TM*
- Semantics of transducers could be dynamic or buffered I/O; dynamic = interactive

---

Finite transducers

- A *Mealy machine* is a 6-tuple \( \langle Q, \Sigma, \Gamma, \delta, \text{out}, q_0 \rangle \) where
  - \( Q \) is a set of states, \( q_0 \) is start state
  - \( \Sigma \) is a finite input alphabet
  - \( \Gamma \) is a finite output alphabet
  - \( \delta \) is a transition function \( Q \times \Sigma \rightarrow Q \)
  - \( \text{out} \) is an output function \( Q \times \Sigma \rightarrow \Gamma \)
- There is no final state because the transducer does not halt
- *Stream language* of a MM is a set of streams \( SL \subseteq (\Sigma \times \Gamma)^\infty \)
Models of interactive and parallel computation

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 such as ICs, controllers, etc.
- A Mealy machine computes a function $\Sigma^\infty \to \Gamma^\infty$ where:
  - $\Sigma^\infty = \{ ax \mid a \in \Sigma, x \in \Sigma^\infty \}$
  - $\Gamma^\infty = \{ ax \mid a \in \Gamma, x \in \Gamma^\infty \}$
- $\Sigma^\infty$ is the set of streams over $\Sigma$

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
Communication

- **One-way communication** is the sending of strings, over a finite alphabet of symbols, from one entity to another.

- **Two-way communication** is the concurrent activity of two entities engaged in one-way communication with each other.

  - Two-way communication does not assume that either entity waits for an input string before emitting output, or that either entity has an exclusive communication relationship 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.

- Computing entity *interacts synchronously* with environment $E$ if $A$ interacts with $E$ and both $A$ and $E$ wait for only one input token before emitting an output token.

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

(Synchronous) sequential interactive computation:
Interaction involving two participants, at least one of which is a finite computing agent (machine, device).

- 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

Persistent Turing Machines

- A minimal extension of TMs expressing sequential interactive behavior (Goldin, et al)
- A PTM is a 3-tape TM with
  - I/O as dynamically generated streams of interleaved inputs and outputs
  - TM executions (macrosteps) iterated
  - A persistent worktape, called a memory, preserved between macrosteps

- Example: automatic car
**Example: answering machine**

- An *answering machine* $A$ is a PTM whose worktape contains a sequence of recorded messages and whose operations are *record message*, *playback*, and *erase*.
- Its TM-computable function $f_A$ is:
  \[
  f_A(\text{record } Y, X) = (\text{ok}, XY) \\
  f_A(\text{playback}, X) = (X, X) \\
  f_A(\text{erase}, X) = (\text{done}, \epsilon)
  \]
- For input stream $(\text{record } m_1, \text{erase}, \text{record } m_2 m_3)$, $A$ generates $(\text{ok}, \text{done}, \text{ok}, \text{ok}, m_2 m_3)$

(Goldin, 1999)

---

**Stream behavior of PTMs**

- The *persistent stream language* (PSL) of a PTM is the set of streams $L \subseteq (\Sigma^* \times \Sigma^*)^\infty$ observable on it
- The set of all I/O streams over alphabet $\Sigma$:
  \[(\Sigma^* \times \Sigma^*)^\infty = \{ (a, x) \mid a \in (\Sigma^* \times \Sigma^*), x \in (\Sigma^* \times \Sigma^*)^\infty \}\]
- PSL is the set of all persistent stream languages
- *Amnesic* PTMs do not make use of their memory, i.e., are equivalent to TMs in that sense
- **ASL**: The set of *amnesic* stream languages
- **Theorem**: $\text{ASL} \subset \text{PSL}$ (Goldin, Smolka et al, 2004), hence PTMs are more expressive than TMs
Stream language and equivalence

- Interactive stream language (ISL) of a PTM $M$: the set of interaction streams $(i_1, o_1), (i_2, o_3), \ldots i_n, o_n \in \Sigma^*$, in which for every $k$, there are memories $w, w'$ such that $f_M(w, i_k) = (o_k, w')$
- Two PTMs are stream equivalent (observationally equivalent) iff $ISL(M_1) = ISL(M_2)$
- 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 ISL
- Other equivalences: bisimilarity, isomorphism

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 and may or may not synchronize with their environments
- To model autonomous agents, standard UML must be extended (Bauer, Muller, Odell, 2000)
3. Multi-stream and indirect interaction

- *Multi-stream interaction* occurs when an entity is concurrently interacting with more than one other entity.

- Let $A$ and $E$ interact asynchronously. If $E$ may be decomposed into $E'$ and $B$, where $E' = E - \{B\}$, then $A$ and $B$ interact indirectly via $E$ iff mutual causality holds between the behaviors of $A$ and $B$.

Multi-stream interaction

- In contrast to sequential interaction, multi-stream interaction may feature:
  - *Nondeterminism* when attempts to write collide
  - *Dynamic linking* and unlinking, creation/destruction of nodes
  - *Indirect* interaction via a shared environment
Direct and indirect interaction

**Direct interaction:**
interaction via messages, where the identifier of the recipient is specified in a message.

**Indirect interaction:**
interaction via persistent, observable changes to a common environment; recipients are any agents that will observe these changes.

- Sequential interaction is direct
- Preconditions for indirect interaction:
  - Agents share access to parts of the environment
  - Persistence of environment
- *Example of indirect interaction:* use of semaphores in process synchronization (critical section problem)

Ubiquity of indirect interaction

- **Social biology:** Social insects interact by modifying common structures or through pheromones
- **Operating systems:** Processes communicate via semaphores in shared memory
- **Coordination languages:** Shared tuple spaces enable coordination in Linda
- **Anatomy:** Cells exchange information via hormones in the blood stream
- **Economics:** A market is an environment for buyers and sellers that serves as a medium for indirect interaction
### Properties of indirect interaction

- **Time decoupling (asynchrony):** State changes persist
- **Anonymity:** Recipient ID not used in access
- **Space decoupling:** Agents need not meet
- **Non-intentionality:** Agents need not have goal of communicating
- **Hybrid nature:** Physical environment may play role
- **Late binding** of recipient

### The message-passing model of concurrency

- Due to Robin Milner: CCS, \( \pi \) Calculus; associated with theory of concurrency and with process algebra
- These models capture the notion of direct interaction by message passing
- Axiom of concurrency theory:
  \[ \text{interaction} = \text{message passing} \]
  i.e., atomic communication of a message from one process to another (targeted send/receive)
- Shared variables are deemed processes
Mission of a multi-agent system

- Algebra and propositional logic provide rules for *evaluation* of formulas
- Algorithms compute recursively definable *functions*
- Sequential-interactive agents offer *services*
- Multi-agent systems accomplish *missions* requiring *quality of service* for all users
- Interaction in MASs may be *asynchronous*
- Mission may require a minimum QoS regardless of number of users (*scalability*)

Indirect interaction and multi-agent systems

- In a MAS characterized by *locality* of interaction and *mobility* of agents, it is only possible for agents to influence overall system behavior by use of indirect interaction
- Richness of multiagent interaction:
  - It is due partly to ability of each agent to interact with multiple others
  - Hence each agent interacts indirectly with *all* others (otherwise system partitions)
Limitations of the message-passing model

- Message passing does not support properties of indirect interaction: anonymity, asynchrony, space decoupling, non-intentionality, and late binding
- Embedded and situated systems aren’t supported
- Suppose agents $A$ and $B$ communicate via shared variable $X$
  - The message-passing model accounts for $A \leftrightarrow X$ and $B \leftrightarrow X$ interaction.
  - …but not between $A$ and $B$ via $X$

Decentralized, self-organizing systems

- Decentralized and self-organizing systems lend themselves to flexibility and adaptiveness
- Where required: in environments that are dynamic, persistent, multi-agent, decentralized, and self-organizing.

Decentralized system: a multi-agent system whose components do not respond to commands from an active director or manager component, and do not execute prespecified synchronized roles under a design or plan.

Self-organizing system: a multi-agent system with a coherent global structure or pattern shaped by local interactions among components, rather than by external forces.
4. Models of parallel and distributed computation

• *Circuit model:* Processing is hard-wired into a combinational circuit that executes an algorithm
• *PRAM* (Parallel Random-Access Machine): an extension of the Von Neumann architecture to multiple CPUs sharing memory

Food foraging problem for ants

• Food is scattered randomly
• Task is to take it to the nest
• Ants are small and limited in intelligence and communicating power
• Food may appear or disappear dynamically
• A solution:
  – Ants walk semi-randomly dropping pheromone
  – Ants tend also to follow pheromone trails
  – Ants carrying food drop special pheromone
  – Trails evolve toward short paths between nest and food
Stigmergy in nature

1. Ants foraging (see slide 4)
2. Termites gathering chips into pile:
   Move at random, pick up chip when encountered, put down when another chip found; the pile structure is used to coordinate creation of pile (StarLogo)
3. Slime mold dividing and aggregating:
   These amoeba may aggregate by emitting a chemical, migrating toward its greatest concentration

Q: Is stigmergy essential for some missions?

PRAM model

- Assume all processors have access to shared memory in O(1) time
- Each processor also has private memory
- Shared memory is globally addressable
- PRAMs are {concurrent, exclusive} read, {concurrent, exclusive} write – CRCW, CREW, ERCW, EREW
- PRAM is the standard assumption for theoretical work in parallel computation
Models of interactive and parallel computation

**Single and multiple instructions and data**

**MIMD**
- Most common
- Every processor may execute a different instruction stream on a different data stream
- May be synchronous or asynchronous, deterministic or nondeterministic

**SIMD**
- All CPUs synchronously execute the same instruction, but on different data items
- Varieties: processor arrays, vector pipelines
- Deterministic

**Shared-memory architectures**
- All CPUs access all memory as global address space
- Memory access may be uniform (UMA) or non-uniform (NUMA)
- Where processors use cache, *cache coherency* may be an issue
Distributed-memory and hybrid architectures

- Memory addresses in distributed-memory systems do not map between CPUs; no global address space exists

- Cache coherency does not apply
- Scalability is in thousands of CPUs
- Hybrid distributed-shared memory systems network multiple symmetric multiprocessors

Parallel programming models

- Programming model is an abstraction lying above hardware and memory architecture and is not architecture specific
- Shared memory
  - May be implemented using distributed physical memory as virtual memory
  - Programming interface is simple, using global addresses
- Threads
  - Like subroutines, called in parallel, sharing some memory
  - UNIX (POSIX) and Java support threads
Other parallel programming models

- **Message passing**
  - Tasks exchange data on network via messages
  - MPI (1994) is de facto industry standard
  - MPI may be used in shared-memory architectures
- **Data parallel**: parallel tasks work on different parts of a common data structure
- **Hybrid**: combines the four above programming models, e.g., MPI with threads or shared memory

5. Models of learning agents

**Learning**

- An agent *learns* from experience $E$, w.r.t. task set $T$, using performance measure $P$, iff:
  - Performance measure under $P$ on tasks in $T$ improves with experience $E$
- **Example**: Driving
  - $T$: driving using vision sensors
  - $P$: average distance without error
  - $E$: images paired with steering commands
Training

- **Assumption:** Distribution of training examples matches distribution of test examples
- **Learning task:** Generalize from training examples (or past experience) to state-perspective action function

Concept learning

- **Domain:** a set, $X$, of instances of a concept
- **Target concept:** $c : X \to \{ T, F \}$
- **Training examples:** a set of pairs $(x, y)$, with $x \in X, y = c(x)$
  - Where $y = F$, example is “negative”
  - Where $y = T$, example is “positive”
- **Goal of learning:** to find $c$ by generalizing from training examples
- **Method:** Search for a hypothesis $h$ s.t. $h(x) = c(x)$ for all $x \in X$
Neural networks

- Human brain: $10^{11}$ neurons, each connected to average of $10^4$ others. Max. switching time: $10^{-3}$ sec
- Each neuron “fires” (output pulse) if summed inputs exceed a threshold
- Artificial NN ANVINN drove car at 70 mph, 1993
- Good for problems with noisy, complex sensory information (cameras, microphones)
- Most common algorithm: Backpropagation
- Slow learning, fast application
- Hard for humans to understand correspondence between problem and NN solution

Decision tree learning

- Classify instances of concepts in trees whose leaves are classifications, e.g., yes/no
- Branches test for attributes
- Robust to errors unlike other concept-learning algorithms
Learning agents

- In learning, percepts improve agent’s future success in interaction
- Components:
  - Learning element (improves policy)
  - Performance element (executes policy)
  - Critic: Applies fixed performance measure
  - Problem generator: Suggests experimental actions that will provide information to learning element

RN95, pp. 525ff

Passive ADP learning

- Adaptive dynamic programming
- Requires transition model, containing information about structure of environment
- Utility $U$ of state $i$ is computed from state’s reward $R$ and utilities of all adjacent states, weighted by transition probabilities:

$$U(i) = R(i) + \sum_j M_{ij}U(j)$$

RN95, p. 603
Temporal difference (TD) learning

- Uses observed transitions and differences between utilities of successive states to adjust utility estimates
- Update rule based on transition from state $i$ to $j$:
  \[ U(i) \leftarrow U(i) + \alpha (R(i) + U(j) - U(i)) \]
  where
  - $U$ is utility,
  - $R$ is reward
  - $\alpha$ is learning rate

Model-based vs. model-free learning

- Transition model is used in adaptive dynamic programming (ADP) learners
- Use of models is associated with knowledge-based AI
- Temporal difference (TD) learning does not use model
- Q learning is model free

RN95, pp. 612-615
Q-values

• Definition: Q-values are values $Q(a, i)$ of expected utility associated with a given action in a given state

• Utility of state:
  $U(i) = \max_a Q(a, i)$

• Q-values permit decision making without a transition model

• Q-values are directly learnable from reward percepts

TDL-based Q-learning agent

Agent attributes:

$Q = $ table of (action, state) values
$N = $ number of previous (action, state) visits
$R[s] = $ reward for state $s$
$a = $ previous action
$s = $ previous state
$s' = $ current state
$\alpha = $ learning rate
$f = $ exploratory function

Algorithm:

$N[a, s] \leftarrow N[a, s] + 1$
$Q[a, s] \leftarrow Q[a, s] + \alpha(R[s] + \max_{a'} Q[a', s'] - Q[a, s])$
$s \leftarrow s'$
return $\arg\max_{a'} f(Q[a', s], N[a', s'])$

RN95, p. 614
Explicit vs. implicit representation

- Learned function may be represented as table (explicit)
- Policies for games with $10^{120}$ states have been represented as weights in a linear function of board features (implicit)
- Implicit representation allows generalization from states visited to those not

RN95, pp. 615-616

Value function methods vs. evolutionary methods of RL

- Evolutionary methods evaluated a policy that is fixed over many test sequences
- Value function methods update policy during execution
- Value function update benefits from information obtained during interaction

SB98, p. 13
Policy learning overview

- “State signal” represents percept, which contributes to state knowledge
- Policy maps from states to actions
- State value is computed using accumulated experience
- Learner adapts policy online, using this experience \((LL, LU, RL, RU, target)\)

6. Partially observable Markov decision problems

<table>
<thead>
<tr>
<th>Environment type</th>
<th>Agent type</th>
<th>Deterministic</th>
<th>Stochastic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accessible</td>
<td>Reflex</td>
<td></td>
<td>Solves MDPs</td>
</tr>
<tr>
<td>Inaccessible</td>
<td>Policy-based non-Markov</td>
<td></td>
<td>Solves POMDPs</td>
</tr>
</tbody>
</table>
Overview

• POMDPs are problems in inaccessible stochastic environments with Markov property

• *Markov property*: probability that system will be in a given state at next step depends only on current state, not on past history (L. Lipsky)

Accessible vs. inaccessible environments

• In accessible environment, agent percept identifies current state

• Hence if environment is stochastic, probabilities of state transitions depend only on current state, not on past history of interaction

• Markov decision problems (MDPs) are associated with stochastic problems in accessible environments
Stochastic vs. deterministic problems

- Deterministic version of decision problem assures single result of an action
- In stochastic version, an action has a given effect with a probability value
- Example: In a maze, action “move-north” may have probability 0.8 to move north, 0.1 east, 0.1 west

RN95, p. 499

Policy search

- Policy: a mapping from states to actions
- Policy is as opposed to action sequence
- Agents that precompute action sequences cannot respond to new sensory information
- Agent that follows a policy incorporates sensory information about state into action determination

RN95, p. 499-500
Transition models

- Definition: Set of probability values for given state transitions under given actions
- $M_{ij}^a$ denotes probability of transition from state $i$ to state $j$ on action $a$
- Transition model is only needed for stochastic problems

Reward vs. utility

- Reward is obtained immediately upon entering a state
- Utility of a state is expected longterm cumulative reward
- Utility can provide guide to rational decisions

RN95, pp. 502-503
### Policy search in POMDPs

- *Value iteration* algorithm calculates utility of each state, from which optimal actions can be computed.
- *Policy iteration* chooses a policy, calculates utility of state under that policy, then repeats at predecessor states.

RN95, pp. 502-505

### Value of information vs. reward value

- In POMDPs, utility of an action may be influenced by:
  - future reward brought closer by an action
  - future percepts made possible by an action
- Value of information obtained in future percepts must be part of a state-action pair’s utility.

RN95, pp. 501-502
References [Interaction]

Bauer, Muller, Odell, 2000.

References [Interaction]

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


References [Learning]

