

David Keil
Framingham State University

6. Models of interactive computation

1. Sequential interaction
2. Basic models of interaction
3. Persistent Turing machines

Readings: Wegner; Goldin; Keil

David Keil Theory of computing 6. Interaction 1/12 1

Inquiry

- Is *communication* part of *computation*?
- Does concurrency require new computational models?

David Keil Theory of computing 6. Interaction 1/12 2

Objectives

- 6a. Distinguish algorithmic from interactive computation
- 6b. Describe models of sequential interactive computation

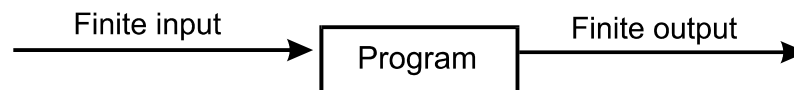
Kinds of concurrency

- *Sequential interaction* alternates input with output
- *Multitasking and threads* may run on one processor
- *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
- **Question: Do Turing machines model these adequately?**

1. Sequential interaction

Algorithmic computation (D. Knuth):

The effective transformation of a finite, pre-specified input, to a finite output, in a finite number of steps.



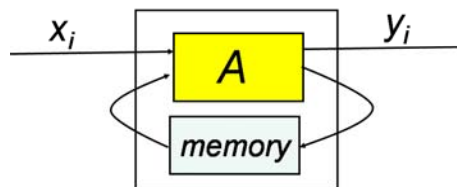
- Algorithms *compute functions*
- A system that executes an algorithm is *closed*

Alternative definition

- *Algorithmic* computation is, equivalently:
 - Halting Turing-machine computation
 - Computation by random-access machine, e.g., in \mathcal{L} language
 - Evaluation of μ -recursive functions
- This definition is derived from the Church-Turing thesis
- Note that this definition excludes all interleaving of input, output, and processing steps

The paradigm shift to interaction

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



Communication

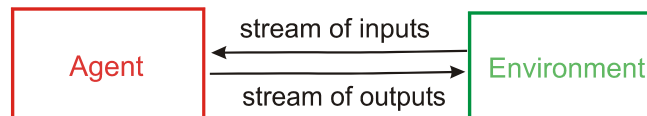
- *One-way communication* is the sending of strings, over a finite alphabet of symbols, from one computing entity to another
- *Two-way communication* is the concurrent activity of two entities engaged in one-way communication with each other
- Two-way communication does not assume that either entity 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 A *interacts synchronously* with environment E if A interacts with E and both A and E wait for input before emitting output.
- *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 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

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
- They may initiate *observation* actions
- To model autonomous agents, standard UML must be extended

Contributions to the theory of interactive computing

- *c*-machine (Turing), finite transducer (Moore)
- Cybernetics: models of feedback systems (Wiener)
- Information theory/communication theory (Shannon)
- Concurrency with message passing: CSP (Hoare), CCS (Milner), π calculus (Milner)
- Recent models of sequential interaction:
I/O Automata (Lynch), Abstract State Machines (Gurevich), Site Machines (van Leeuwen, Wiedermann)
- Interaction Machines and Persistent Turing Machine (Wegner, Goldin)
- *Emerging intuition*: Interaction is part of computation

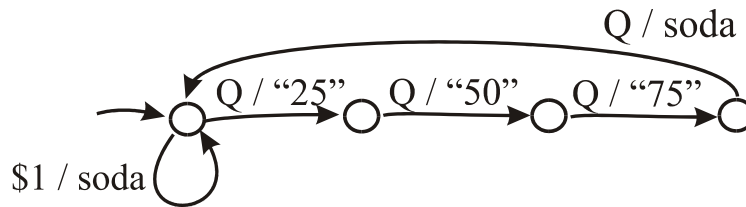
2. Basic models of interaction

- *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

Finite transducers

- A *Mealy machine* is a 6-tuple $\langle Q, \Sigma, \Gamma, \delta, out, q_0 \rangle$ where
 - Q is a set of states, q_0 is start state
 - Σ is a finite input alphabet
 - Γ is a finite output alphabet
 - δ is a transition function $Q \times \Sigma \rightarrow Q$
 - 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 $SL \subseteq (\Sigma \times \Gamma)^\infty$

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.
- Let Σ be input alphabet, let Γ be output alphabet
- 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 $((Q, 25), (Q, 50), (Q, 75), (Q, \text{soda}))$
- I.e.:
 $((\$1, \text{soda}) \mid ((Q, 25), (Q, 50), (Q, 75), (Q, \text{soda})))^\infty$

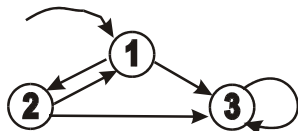
Related ideas

- 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

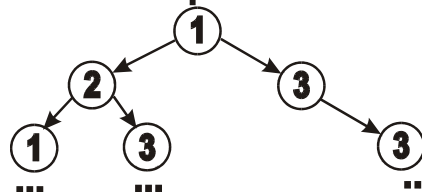
Kripke structure

- 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

Kripke structure

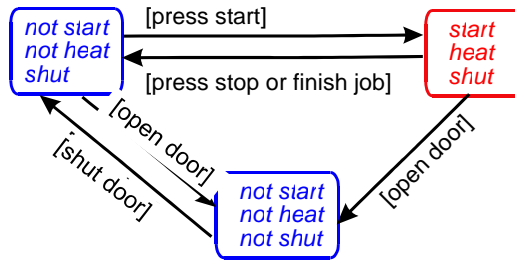


Its computation tree



Computation Tree Logic

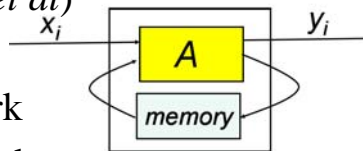
- *State* formulas express assertions about a state (see labels on states of statechart below)
- *Path* formulas express assertions about a set of paths in a computation tree



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

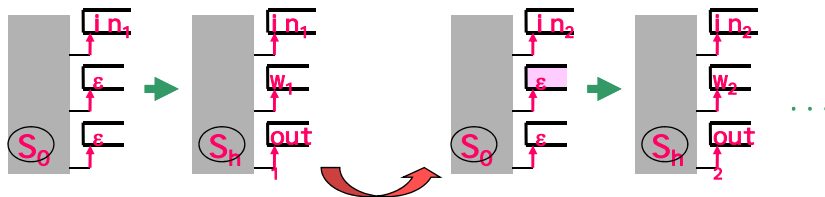
3. Persistent Turing Machines

- A minimal extension of TMs expressing *sequential interactive behavior* (Goldin, *et al*)
- A *PTM* is a 3-tape TM with
 - Tapes for input, output, work
 - I/O as dynamically generated *streams of interleaved inputs and outputs*
 - TM executions (*macrosteps*) iterated
 - A persistent worktape, called a *memory*,
- *Example:* automatic car



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, produces and output, and updates the work tape



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:

$$f_A(\text{record } Y, X) = (\text{ok}, XY)$$

$$f_A(\text{playback}, X) = (X, X)$$

$$f_A(\text{erase}, X) = (\text{done}, \lambda)$$
- For input stream that begins (record m_1 , erase, record m_2 , playback), A generates (ok, done, ok, m_2), where m_1, m_2 are messages

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 Σ :
 $(\Sigma^* \times \Sigma^*)^\infty = \{ (a, x) \mid a \in (\Sigma^* \times \Sigma^*), x \in (\Sigma^* \times \Sigma^*)^\infty \}$
- Persistent stream language (PSL) of a PTM M :
the set of interaction streams $(i_1, o_1), (i_2, o_3), \dots$
where $i_n, o_n \in \Sigma^*$, in which for every k , there are memories w, w' such that $f_M(w, i_k) = (o_k, w')$
- \mathcal{PSL} is the set of all persistent stream languages

Stream languages and equivalence

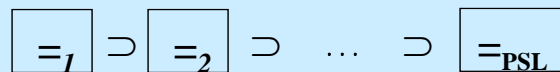
- Two PTMs are *stream equivalent* (*observationally equivalent*) iff $PSL(M_1) = PSL(M_2)$
- Two memories of M , w_1 and w_2 are equivalent iff sub-PTMs with w_1 and w_2 as starting memories have the same PSL
- *Isomorphism*: the states have a structure-preserving bijection; design and behavior are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

Amnesic PTMs

- *Definition:* PTMs that do not use their work tapes
- *Example:* squaring machine ($\text{out}_i = \text{in}_i^2$) are “half-way” between TMs and PTMs.
- APTMs extend TMs with stream-based semantics, do not make use of their memory, i.e., are equivalent to TMs in that sense
- Unlike PTMs, they lack persistence.
- *ASL:* The set of *amnesic* stream languages, i.e., languages of amnesic PTMs
- In a PSL, order matters; in an ASL, it doesn't, because no macrostep is affected by previous ones

Infinite Equivalence Hierarchy

- *Theorem:* $ASL \subset PSL$ (Goldin, Smolka et al, 2004), hence **PTMs are more expressive than TMs**
- $L_k(M)$ = **k -stream prefix language of PTM M :** the set of prefixes of length $\leq k$ for streams in $PSL(M)$.
- Corresponding **notion of equivalence:** $M_1 =_k M_2$ is defined as $L_k(M_1) = L_k(M_2)$
- Below are strict inclusions of equivalence classes



(D. Goldin, 2004)

Expressiveness of PTMS

Two ways to show that PTMs are more expressive than Amnesic PTMs (and, by extension, TMs):

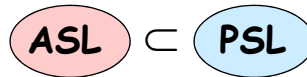
1. Collapse of the equivalence hierarchy.

$$\boxed{=}_1 = \boxed{=}_{PSL}$$

2. Smaller set of stream languages.

ASL = {PSL(M): M is an amnesic PTM}

PSL = {PSL(M): M is a PTM}



(D. Goldin, 2004)

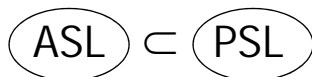
Summary of Results [I&C'04]

$$\boxed{=}_{ASL}$$

=

$$\boxed{=}_1 \supseteq \boxed{=}_2 \cdots \boxed{=}_{\infty} \supseteq \boxed{=}_{PSL}$$

$$\boxed{=}_{ms}$$



(D. Goldin, 2004)

Concepts

algorithm	finite transducer
amnesic stream	interaction
language	Mealy machine
asynchrony	persistent stream
autonomous agent	language
communication	Persistent Turing
concurrency	Machine
direct interaction	sequential interaction
	stream I/O
	synchrony

References

- Bauer, Muller, Odell, 2000. UML.
- Dina Goldin. Persistent Turing Machines as a Model of Interactive Computation. FoIKS'2000, *LNCS 1762*, pages 116-135, 2000.
- D. Goldin, S. A. Smolka, P. Attie, and E. Sonderegger. Turing Machines, Transition Systems, and Interaction. *Information and Computation* 194(2): 101-128, 2004.
- David Keil and Dina Goldin. Adaptation and Evolution in Dynamic Persistent Environments. In *Proc. FInCo2005*.
- David Keil and Dina Goldin. Modeling Indirect Interaction in Open Computational Systems. *TAPOCS Workshop, Proc. WETICE 03*.
- Peter Wegner. Why interaction is more powerful than algorithms. *CACM* 40 (5), 1997.

References