Res Publica: The Universal Model of Computation

Nachum Dershowitz

School of Computer Science, Tel Aviv University, Ramat Aviv, Israel

Abstract

We proffer a model of computation that encompasses a broad variety of contemporary generic models, such as cellular automata—including dynamic ones, and abstract state machines—incorporating, as they do, interaction and parallelism. We ponder what it means for such an intertwined system to be effective and note that the suggested framework is ideal for representing continuous-time and asynchronous systems.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Models of computation, cellular automata, abstract state machines, causal dynamics, interaction

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

> The nature of the process is truly characterized by Glaucon, when he describes himself as a companion who is not good for much in an investigation, but can see what he is shown, and may, perhaps, give the answer to a question more fluently than another.
> —Plato, *The Republic*

1 Purpose

The goal of this study is to design a model of computation that encompasses various and sundry generic models, such as dynamic cellular automata [1], as well as interactive and parallel abstract state machines [2, 3]. Furthermore, the model should be capable of dealing with continuous-time and asynchronous systems.

We employ a political metaphor.

2 The State Model

Blocs. A *bloc* is an interconnected collection of *states* that evolve over time. The number of states in a bloc may be finite or infinite. States communicate with each other via (communication) *channels*. Not only do the internals of states evolve, but their connections may be reorganized. Furthermore, it may be possible for new states to be created and connected to existing ones.

Maps. We draw channels as pipes (looking like hoses) emanating from the client state (on the requesting end) and connected to the serving state (which owns the data that is being made public). A serving state may allow client to update sections of the shared data. Arrows along the channel can be used to indicate that data flows along a channel in one direction only.

Computer Science Logic.
Editor: Simona Ronchi Della Rocca; pp. 1–5

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

States. Each state is a logical structure (consisting of a domain, first-order vocabulary, and interpretations for the operators) whose evolution is governed by its native *policy*—which may be *natural* (fixed by laws), *algorithmic* (dictated by a program), or *arbitrary* (controlled by some external agency)—and may react to its environment. As such, a state contains interpretations for the *functions* in its vocabulary (constants may be viewed as scalar functions and relations as truth-valued functions). Only the interpretations given by a state to its functions may change during evolution; the domain and vocabulary are fixed throughout.

Domains. All states in a bloc share the same domain, but can have different vocabularies. Domains may be finite (*automata*), countably infinite (*machines*), or uncountable (*processes*).

Names. States have (unique) identifying *names*, taken from a *namespace* that is included in the domains of states. Pipes in a graphical representation of this model of computation depict the use of names.

Resources. A subset of each state's vocabulary are designated *public*. Their values are made visible to other states; *private* functions are not. A *resource* is a (named) state along with one of its public functions. One can consider a framework in which public resources can be accessed but not modified by others; think of them as (read-only) *communiqués*. Alternatively, some resources can be designated *shared* and allow for modification by foreign states. No bound is placed on the number of channels connected to a state or the number of shared resources.

Assets and Agents. From the point of view of the client of a resource, a shared resource to which it is connected is its *agent*, while a public resource that is not modifiable is an *asset* of its.

Vassals. A state can only modify the values of its own functions or of shared resources to which it has access. To provide differential access to its public data, a state can set up *vassals* (or "satellites"), each of which connects to it by a private one-way channel, keeping the name of the controlling state secret (not publicly available). The vassal state can continuously retrieve the relevant part of the data from its master state and pass it on to whichever states are connected to it, the vassal.

Realignments. The topology of a bloc can change due to modifications of (the values of) its channels. In particular, if the value of a resource is itself a name, then a state can change an outgoing channel to refer to the state named by the resource.

Locations. *Locations* in a state are determined by function symbols (from the vocabulary) and domain values for its arguments (as per the arity of the symbol); it is the contents of locations that change when an interpretation is updated.

Puppets. A state may also create a *puppet*, which is a state with the same domain and vocabulary, running the same policy. Before releasing the puppet to run on its own, the controlling state may set various values in the puppet; all other locations in the puppet will retain their default values.

3 State Evolution

Time. States evolve over time, where *time* $\mathbb{T}$, in general, can be any linearly-ordered domain, with ordering $\leq$ and minimal element, denoted 0. Let $\mathbb{S}$ be the initial intervals $[0, t)$ for all $t \in \mathbb{T}$.

Discrete Time. For discrete systems, time is the natural numbers $\mathbb{N}$, with initial segments $\mathbb{S} = [0..n)$, for $n \in \mathbb{N}$.

Continuous Time. For continuous behavior, time $\mathbb{T}$ would be the non-negative reals.

Signals. Each resource to which a state is connected provides it with a *signal*, which is a function from an interval in $\mathbb{S}$ to the domain of the bloc. A signal defined for an interval $[0, t)$ has *length* t. Concatenation of a signal of length s with one of length t gives a signal of length $s + t$ in the obvious way.

Interaction. Channels provide a means for communication between states, but there is no special mechanism for explicitly responding to requests. Clearly, the signal emitted by one resource may depend on signals emitted by others. That is the nature of interaction.

Environments. The ensemble of signals reaching a state constitutes its *environment*. Let the possible environments, Σ, be all tuples of signals of the same length. The *width* of an environment the number of components in the tuple. The concatenation $\alpha\beta$ of environments $\alpha, \beta \in \Sigma$ of the same width is the tuple of concatenated signals. Write $\alpha \leq \gamma$ if there exists a β such that $\alpha\beta = \gamma$.

Evolutions. Policies are described by *transition* functions τ (perhaps multivalued) that map states and environments to states. That is, $\tau : X \times \Sigma \rightrightarrows X$. The *evolution* of a state x for a given environment γ is the sequence of states obtained in this way: $\{\tau_\alpha(x)\}_{\alpha \leq \gamma}$.

Causality. Let τ be a transition function. Transitions must be *causal* ("retrospective"), depending only on the past, so that $\tau_{\alpha\beta}(x) = \tau_\beta(\tau_\alpha(x))$ for all states x, where $\alpha\beta$ is a concatenated environment. If τ is multivalued, then τ_β should be understood as extended to sets. Put differently, $\tau_{\alpha\beta} = \tau_\alpha \circ \tau_\beta$, as relations.

Federations. One can view a subset of the states as one *federated* state. The transitions of the federation depend on its external environment, mediated by channels from the outside.

Globe. The *global* federation consists of the totality of states, or at least those states that are governed by programs or processes.

4 State Policies

Programs. Algorithmic transitions may be described by programs. Programs may be expressed in the basic language of abstract state machines [6], which includes the following at a minimum:

- general assignments: $f(s_1, \ldots, s_k) := t$ (terms s_i, t in the vocabulary of the state)
- conditionals: **if** c **then** P (Boolean term c and program P), and
- parallel composition: $P \parallel Q$ (programs P, Q).

In addition, we want

- higher-order assignments: $f := g$, where f and g are functions (of the state vocabulary) of the same arity, and
- serial composition: $P; Q$ (programs P, Q).

Channels. A channel is a name-valued location. A *foreign* location is indicated by an expression of the form $p.\ell$, where p is a channel and ℓ is a location. Only local locations and shared resources may appear on the left of assignments. A foreign resource on the left of an assignment is an agent; if it only appears on the right side or in conditions, it is an asset (that is read-only).

Dependence. A new state may be conceived with a

- creation assignment: $p :=$ **new** $f, g, \ldots$ **allow** $h, k, \ldots$.

The new puppet state, pointed to by p, with have public functions $f, g, \ldots, h, k, \ldots$, with the second half of the list shared freely. When launched, the puppet will run the same programmed policy as its parent. Assignments may be made to locations in unlaunched puppets (high-level assignments are of help here); flags can be used to specialize the behavior of puppets.

Independence. The

- launch command: **free** p

activates the program in the puppet pointed to by p, at which point the parent can no longer modify it on its own. The puppet is now independent.

Federations. The program of the federation as a whole is just the union of the programs of its constituent states, with functions disambiguated by the name of the state they reside in. (Of course, some states might not be governed by programs, but rather provide measurements of natural phenomena like temperature and barometric pressure.) Whereas an individual programmed state has a bounded number of channels it owns, a federation can create more and more new states, each of which is connected to non-federated states.

Flows and Jumps. A *jump* in the evolution of a continuous-time state is a change in its dynamics, in contrast with *flows*, during which the dynamics are fixed. See [5].

Flows. For continuous-time systems, the discrete programming language is extended with

- continuous (explicit) assignments: $f(s_1, \ldots, s_k) :\approx t$,

which stay in force until a new assignment is made to the same *term* by some program.

Jumps. Jumps are effected by conditionals. Additional constraints on algorithmic evolution make sense in the continuous context. These include that tests should test for conditions that have non-zero duration and that the dynamics of a system change only finitely often in a finite period of time.

Conflicts. Programs as described above can cause conflicts ("clashes") when different (discrete or continuous) assignments (in one or more state programs) attempt to assign different values (at the same moment) to a single location. The outcome of any such conflict is any one of the possibilities. (These nondeterministic semantics are preferable to a system crash.)

Continuity. Continuous assignments may involve infinitesimal time, dt, provided the outcome is independent of the choice for dt. This is a continuity requirement of sorts. One can conceive of implicit specifications of continuous behavior, as well.

5 State Policies

Clocks. To achieve synchronous behavior in a continuous-time environment, there would need to be a global clock to which other states are connected, directly or indirectly.

Archives. When foreign locations provide only read-only resources, write abilities to a public (but not shared) memory need to be achieved via requests—as in modern hardware. A state p can allocate resources for requests r, addresses a, and values v, which it makes available to a memory module. The latter runs a program of the sort **if** $p.r$ **then** $m(p.a) := p.v$, for some "storage" function m. A similar setup may be used to serve stored values.

Queues. When unboundedly many states use the same controlled archive, some queueing mechanism needs to be set up, by means of which individual states can place requests while the archive deals with them one at a time.

Data. If (automata) states share a finite domain (as in cellular models [1]), then unbounded memory is achievable by means an unbounded number of connected states, in which case an unbounded number of steps may be needed to reach a particular datum.

Interfaces. To model a physical or biological system in which units are each governed by rules, but adjacent units exchange values or signals, one could represent their interface as a channel. For example, the temperature of a wall would be a public function over $\mathbb{R}^2$ of one side or the other.

Effectiveness. In general, for a system to be deemed *effective*, not only should its transitions and evolutions be describable by a finite text, but also the initial states with the operations they are endowed with. For a bloc to be effective, it should have finitely many states, each governed by an effective algorithm [4]. The number of states and their inter-connections may grow unboundedly during its evolution.

6 Conclusion

We believe that most of the usual and unusual models of computation are instances of this paradigm.

References

1 Pablo Arrighi and Gilles Dowek, July 2012, "Causal graph dynamics", *Proceedings of the 39th International Colloquium on Automata, Languages, and Programming (ICALP 2012)*, Warwick, UK, Lecture Notes in Computer Science, vol. 7392, Part II, pp. 54–66. Available at `http://arxiv.org/pdf/1202.1098v3` (viewed July 10, 2013).

2 Andreas Blass and Yuri Gurevich, 2006, "Ordinary interactive small-step algorithms, Part I". *ACM Transactions on Computational Logic*, 7(2), pp. 363–419. Available at `http://tocl.acm.org/accepted/blass04.ps` (viewed July 10, 2013).

3 Andreas Blass and Yuri Gurevich, June 2008, "Abstract state machines capture parallel algorithms: Correction and extension", *ACM Transactions on Computation Logic*, 9(3), Article 19. Available at `http://research.microsoft.com/en-us/um/people/gurevich/Opera/157-2.pdf` (viewed July 10, 2013).

4 Udi Boker and Nachum Dershowitz, August 2010, "Three paths to effectiveness", in Andreas Blass, Nachum Dershowitz, and Wolfgang Reisig, editors, *Fields of Logic and Computation: Essays Dedicated to Yuri Gurevich on the Occasion of His 70th Birthday*, volume 6300 of *Lecture Notes in Computer Science*, pp. 36–47, Springer, Berlin. Available at `http://nachum.org/papers/ThreePathsToEffectiveness.pdf` (viewed July 10, 2013).

5 Olivier Bournez, Nachum Dershowitz, and Evgenia Falkovich, May 2012, "Towards an axiomatization of simple analog algorithms", in Manindra Agrawal, S. Barry Cooper, and Angsheng Li, editors, *Proceedings of the 9th Annual Conference on Theory and Applications of Models of Computation (TAMC 2012, Beijing, China)*, volume 7287 of *Lecture Notes in Computer Science*, pp. 525–536. Springer, Berlin. Available at `http://nachum.org/papers/SimpleAnalog.pdf` (viewed July 11, 2012).

6 Yuri Gurevich, 1995, "Evolving algebras 1993: Lipari guide", in Egon Börger, editor, *Specification and Validation Methods*, pp. 9–36. Oxford University Press. Available at `http://research.microsoft.com/~gurevich/opera/103.pdf` (viewed July 10, 2012).