An Ontology of States

Andrew Polonsky[1] and Henk Barendregt[1,2]

[1] Institute for Computing and Information Sciences (iCIS),
Radboud University Nijmegen, The Netherlands
[2] Netherlands Institute for Advanced Study, Wassenaar

1 Introduction

The notion of state is ubiquitous in analysis of computational systems. State introduces intensional content into a dynamical process which cannot be directly observed from outside. Without a state, the process is defined purely by its input-output behaviour, and is thus expected to run itself out toward a final result, ie, compute some function. The injunction of internal data that has causal effect on the execution of a system can thus be said to be the step that extends the concept of a function to that of a process, which is no longer guaranteed to terminate.

On the hardware level, this step is observed as one ascends from combinational circuits to sequential circuits. The former concerns logical circuits that produce values that depend only on the given input. The latter refers to circuits that have 'memory': an internal state that depends on the previous history of execution in addition to the input data. The fundamental circuit element that allows for state behavior is the flip-flop, that can store one bit of data in a virtual feedback loop. (This circuit also exhibits another characteristic of systems with memory: the presence of some clock mechanism.) The significance of circuits with state is that they make it possible to construct a fully functional computer system — which requires registers, arithmetic processors, instruction sequencing, storage, etc.

On the software level, this distinction manifests itself as the difference between programs which execute some specific algorithm and then terminate (e.g. compilers, text analysers, theorem provers, database search) and those that interact with their environment while they are running (e.g. text editors, web servers, operating systems). In the latter family, the behavior of the program at a particular moment may be determined not only by the current user input, but by the previous history also, so that there is additional information needed to determine how the program will act at a given instant.

On the level of language design, the presence of intensional data is part of the great divide between the pure functional languages and imperative (also, object-oriented) languages. Because of *referential transparency*, the meaning of an expression in purely functional languages is defined independently of the context or history of execution. In contrast, languages in the latter family also allow destructive operations on data which makes it possible to have data structures whose internal content changes over time.

P. Achten and P. Koopman (Eds.): Plasmeijer Festschrift, LNCS 8106, pp. 18–26, 2013.

Functional languages are very convenient for writing programs of the first kind, where the input data must be transformed in some algorithmically complex way; indeed, the syntax of languages such as `Clean` very closely mirrors the actual mathematical definition of the function being computed. Thus functional programs are much easier to analyze and prove properties of than their imperative counterparts.

However, many real-world applications require programs of the second kind; for this, stateful programming becomes unavoidable (similarly to how purely combinational circuits become inadequate for building complex computational systems). Thus, much of the research in the field of functional programming has been devoted to finding the most clean way of incorporating intensional content into functional programs. The design of the `Clean` language is directly based on the fruits of this research, as is made evident in its very name. In addition to fundamental innovations in language design, this work also introduced ideas in programming language theory of independent mathematical interest, [3] and [4].

The practical possibilities offered by the `Clean`-style uniqueness typing solution to the state problem can be observed in the implementation of the iTasks system, [5] and [6].

In this paper, we offer a meta-level investigation of the notion of a state from the conceptual/logical points of view. We will show that there are several ways with which a state specifies the intensional content of a sytem.

2 Agents and Systems

An *agent* is a system that receives input from the environment and comes into action. Classes of agents are (in historical order)

1. Molecules and molecular machines.
2. Organims, from unicellular ones to homo sapiens.
3. Computer systems, from ad-hoc chips to super-computers.

The simplest way to describe an agent is recording its input-output (I/O) behaviour. This may be done in natural or testing circumstances. This leads to a behaviour function. Denoting the set of possible inputs by I and the set of possible actions by A, we see that such a purely behavioral system is nothing but a map

$$M : I \to A \tag{1}$$

which specifies which actions are to be taken for every input. One may rewrite this as

$$M(i) = a \in A \tag{1'}.$$

In most cases this behaviouristic approach is limited. Nevertheless some agents can be described in this way. Their actions uniquely result from the input, either in a deterministic way or in a non-deterministic way.

More interesting agents may react differently under the same circumstances. This is the reason that the notion of state is important, as the model (1) is inadequate.

3 What Is a State?

Observing an agent we may tentatively say that its behaviour depends on a state. The behaviour function now becomes

$$M : I \times S \to A. \tag{2}$$

One also may write

$$M(i, s) = a \in A. \tag{2'}$$

But do these states exist? To a biologist a state that fully determines together with the input the action may seem doubtful. Using some mathematical hubris one can nevertheless affirm that states do exist. An agent M_1 at moment t_1 can be said to be in the same state as a similar (having the same classes of I/O) agent M_2 at moment t_2 iff M_1 and M_2 react on an input $i \in I$ with the same action. In particular one has defined now when an agent M is at moment t_1 and t_2 in the same state by taking $M_1 = M_2 = M$. Now with the principle of abstraction, see [1], one can define 'state' from the relation 'to be in the same state':

Definition. A *state* is an equivalence relation consisting of pairs (M, t) all being in the same state.

In this way a state is a higher-order concept: a specific I/O relation.

Like the notion of state of a gas in a closed vat (a vector in a $6 * 10^{23}$ space describing for all (10^{23}) gas molecules its 3 position and 3 momentum coordinates) the state of a biological organism or even of a digital device can never be fully known: the amount of data is over-astronomical. So one may wonder why to introduce states, being what seems to be an example of mathematical hubris. The reason to have states is that one may reason about them.

The ontology for state as determined by (2) is a (deterministic or non-deterministic) map. If $s : I \to A$, then one can interpret the action (2) as

$$M(i, s) = s(i) \in A. \tag{2''}$$

Although this ontology is satisfactory for the states used in (2), there is a need for a more complex notion of state.

4 Turing-Like Machines

It is well known that a classical Turing machine can be described as a triple $\langle \Sigma, Q, \delta \rangle$, where Σ is a set of symbols, with $b \in \Sigma$ is a special element ('blank'), Q is a set of 'states' with $q_0 \in Q$ a special state (start), and finally

$$\delta \colon \Sigma \times Q \rightharpoonup \Sigma \times Q \times \{L, R\},$$

is a partial map. A set of 'final states' is not needed as it can be simulated by states $q \in Q$ such that $\delta(q, -)$ is never defined. There is a two sided infinite linear tape consisting of cells of order type $\mathbb{Z}$ and a read-write head placed on

one of the cells. If the machine is in state q and the head reads a symbol a and $\delta(a, q) = \langle a', q', L|R\rangle$ is defined, then

I the machine jumps to state q' and symbol a is overwritten by a';
II the head moves over the tape to the left or right, depending on whether the last element of the output was an L or an R.

This description can be slightly generalized in such a way that the resulting 'Turing-like' machines describe 'agents' dealing with input/output (I/O) that include robots, animals and even humans in an abstract way. Now the machine is described as a 4-tuple $\langle I, Q, A, \delta\rangle$, where I is a set of inputs, Q is a set of states, A is a set of actions, and finally

$$\delta\colon I \times Q \rightharpoonup A \times Q$$

is a partial map. By taking $I = \sigma$ and $A = \{L, R\} \cup \{W(a) \mid a \in \Sigma\}$, with $W(a)$ having as meaning 'write the symbol a, the classical Turing machine can be seen as a Turing-like machine. But now I also can be seen as information presented from the outside world through sensors, or the inner world part of memory; and A can be seen as actions including movements of the robot, and focussing attention on a part of memory, relevant in the given environmental context.

If we now suppose that every step taken by the agent M depends also on some current state that is preserved between successive cycles of execution, then, letting S denote the set of these states, such a machine is specified by a map

$$M : I \times S \to A \times S \tag{3}$$

Thus M sends the pair (i, s) of an input and a state to a pair (a, s) consisting of the action to be taken as well as the new state the system is put into.

In this model, the state space S acts as a hidden parameter in the specification of the system.

Modelling living beings in this way, one can ask the question whether there does exists something like a state that determines action (and the next state). The extensional apporach is to say that an agent at different moments is in the same state whenever equal inputs deliver equal actions (and new states). So a state is seen as a map transforming input to action (and a possibly new state). This yields the recursive domain equation

$$S \cong (I \to (A \times S)) \tag{3'}$$

5 Solving the Recursive Domain Equation

We now solve the recursive domain equation

$$S \cong (I \to (A \times S)) \tag{4}$$

Since the variable S appears positively on the left side, the initial solution to this equation is an inductive type. Categorically, this universal solution turns out to

be empty, because $\emptyset$ maps initially to every object and satisfies $(A \times \emptyset)^I \cong \emptyset^I \cong \emptyset$. (Unless I itself is empty, in which case the unique solution to (4) is the singleton set consisting of the empty map $\emptyset : \emptyset \to (A \times \{\emptyset\})$.)

It is therefore preferrable to assume that we begin with some initial set of states S_0, and take the closure by the functor $F(X) = (A \times X)^I$. (This solution is universal among all sets containing S_0.)

Explicitly, such a solution is found by infinitely iterating the functor F and taking the direct limit:

$$\begin{aligned}
S_0 &= S_0 \\
S_1 = F(S_0) &= (A \times S_0)^I = A^I \times S_0^I \\
S_2 = F(S_1) &= (A \times (A^I \times S_0^I))^I \\
&\cong A^I \times (A^I)^I \times (S_0^I)^I \\
&\cong A^{I+I^2} \times S_0^{I^2} \\
S_3 = F(S_2) &= (A \times (A^{I+I^2} \times S_0^{I^2}))^I \\
&\cong A^I \times (A^{I+I^2})^I \times (S_0^{I^2})^I \\
&\cong A^I \times A^{I^2+I^3} \times S_0^{I^3} \\
&\cong A^{I+I^2+I^3} \times S_0^{I^3} \\
S_4 = F(S_3) &\cong A^{I+I^2+I^3+I^4} \times S_0^{I^4} \\
&\vdots \\
S_n &= A^{I+\cdots+I^n} \times S_0^{I^n} \\
&\vdots
\end{aligned}$$

The limit of the above sequence is the type

$$S := S_\omega = A^{I^+} \times S_0^{I^\omega}$$

where

$$X^+ := \sum_{n>1} X^n$$

is the set of strings of *positive length* over a set X. It satisfies the equation

$$X^+ \cong X + X \times X^+ \tag{5}$$

On the other hand, X^ω is the set of all sequences (or *streams*) over X, which satisfies the equation

$$X^\omega \cong X \times X^\omega \tag{6}$$

Using (5) and (6), we trivially verify that

$$\begin{aligned} F(S) &= (A \times S)^I \\ &\cong A^I \times S^I \\ &= A^I \times (A^{I^+} \times S_0^{I^\omega})^I \\ &\cong A^I \times (A^{I^+})^I \times (S_0^{I^\omega})^I \\ &\cong A^I \times A^{I \times I^+} \times S_0^{I \times I^\omega} \\ &\cong A^{I + I \times I^+} \times S_0^{I \times I^\omega} \\ &\cong A^{I^+} \times S_0^{I^\omega} \\ &= S \end{aligned}$$

So that $F(S) \cong S$, as desired.

Notice that the inductive process that builds up S is correlated with the time axis of execution: the n'th approximant S_n contains precisely enough data to run the machine for n steps.

Another curiousity of this space of solutions is that it is a product of two function spaces. Indeed, $S = A^{I^+} \times S_0^{I^\omega}$ consists of pairs of maps (f, g), with $f : I^+ \to A$ and $g : I^\omega \to S_0$. We will now analyze the meaning of these constituent functions.

We note that the first factor, specifying a function $f : I^+ \to A$, serves to determine which action is taken by M after some finite sequence of inputs $i = (i_1, \ldots, i_n)$. This corresponds to the *extensional* part of the specification of M, as every value of this function can be observed by feeding M a required string of inputs. Note that it has no relation to the initial set S_0.

The second factor, on the other hand, declares a function $g : I^\omega \to S_0$. It is curious that no value of g can be known after finitely many inputs. Rather, g may be interpreted by stipulating that, given an *infinite* sequence $x = (x_n)$ of inputs (running the system 'to the end of time'), M ultimately comes to some 'state' $s_x \in S_0$, which ascribes it intrinsic identity that cannot be measured by any actions it takes. In other words, g is the *intensional* part of the specification of M.

By taking $S_0 := 1 = \{0\}$ to be a singleton, we find the space of purely extensional solutions, where the second component is projected away: $S_e = A^{I^+} \times 1^{I^\omega} \cong A^{I^+}$

If we furthermore stipulate any machine $M \in S$ takes some action $a \in A$ *before* the first input is given, then the space of solutions is

$$S = A \times A^{I^+} \cong A^{1 + I^+} \cong A^{I^*}$$

where I^* is the space of finite strings of non-negative length. This set S is nothing but the set of *A-labelled trees* over I.

In conclusion, every system specified by a map $m : I \times S \to A \times S$ consists of two components: an extensional (or *behavioral*) part, given by an A-labelled

I-branching tree, and an intensional part, that merely stipulates some 'hidden variable' determined only by the whole run of the system to infinity.

6 Solution via Scott Domains

The calculation of the recursive type $F(S) = S$ is somewhat better behaved if we work in the category of Scott domains, see [2]. This is effected by turning every set in question (I, A, S) into a flat cpo by adjoing a bottom element $\perp$ and declaring it to be below every other element.

In this setting, we may find S_0 as a subset of the limit S_ω via the embedding $s \mapsto (\perp_A, c_s)$ which sends $s \in S_0$ to the pair consisting of the bottom action and the constant function with value s. Here S_ω with this embedding is indeed universal among all algebras X for the functor F together with an embedding $S_0 \hookrightarrow X$.

Furthermore, since every cpo has a bottom element, the functor F itself has the universal solution in which S_0 is terminal cpo $\{\perp\}$. Note that this cpo doesn't grow during the iteration $\{\perp\}, \{\perp\}^I, \{\perp\}^{I^2}, \ldots$ So in the limit, the second factor appears as the 1-element cpo, giving the pure extensional solution, which is the initial algebra for the functor F in the category of cpos.

7 Continuous Time

As a final rumination, let us consider how the former analysis could be employed in a continuous setting. The first suggestion could be to take the sets I, S, A to be topological spaces, and consider a continuous map

$$\Phi = (u(i, s), a(i, s)) : I \times S \to S \times A$$

representing evolution of state and action with respect to input and previous state. Since we want the state $s \in S$ to evolve "continuously" with each application of Φ, it seems necessary that, whenever $\Phi(i, s) = (s', a')$, we must have $s = s'$. But then repeated iteration of Φ can never move the state!

The resolution of this dilemma is to consider $\Phi = (u, a)$ as a family of operators of smaller and smaller "clock ticks". The input, state, and action then become functions that depend on time.

Thus the question of realizability of a prescribed behavior using a stateful system takes the following form:

Given functions $i(t) : [0, 1] \to I$ and $b(t) : [0, 1] \to A$, when can we find a space S admitting functions $s : [0, 1] \to S$ and $a : I \times S \to A$ such that

1. $b(t) = a(i(t), s(t))$
2. $s(t + dt)$ depends on $i(t)$ and $s(t)$ in a "continuous way".

In order to expore a more precise formulation of the second condition, let us simplify the analysis by first considering the case where the input is held constant: $i(t) = i_0$. We want to think of the evolution of the state during this

interval as a "continuous application" of some state function $v(s)$. This wish could be attained if we are provided a map $v_0 : S \to S$ together with an infinite collection of "compositional square roots": maps

$$v_n : S \to S, \qquad n \geq 0$$

such that

$$v_{n+1} \circ v_{n+1} = v_n$$

We then define v_t for every $t \in [0, 1]$ by prescribing its values on the dyadic rationals: for $t = \frac{t_1}{2} + \frac{t_2}{4} + \cdots + \frac{t_n}{2^n}$, put

$$v(s, t) = v_1^{t_1} \circ \cdots \circ v_n^{t_n}(s)$$

This would define a continuous map v from $S \times [0, 1]$ to S if we can provide, for each $s \in S$, that

$$\lim_{n\to\infty} v_n(s) \quad \text{exists}$$

and that the value of this limit varies continuously with s.

In fact, it must then follow that the limit above is actually equal to s, capturing the intuition that $v(s, t)$ represents infinitesimal evolution of $u(i, s)$.

Now, given $s_0 \in S$, we can define $s : [0, 1] \to S$ by

$$s(t) = \begin{cases} s_0 & t = 0 \\ v(s_0, t) & t > 0 \end{cases}$$

Thus, the right way to ask the question of what is the "next value" of Φ is to consider the infinitesimal change in the state after an infinitesimal tick of time. This naturally leads to the question of the derivative of s.

In fact, in order to accomodate the possibility of changing input, passing to the derivative cannot be avoided, because the small changes in the value of the input must be integrated into the changes of the state from the very beginning.

Now, suppose that the following expression is well-defined for each $s \in S$:

$$\dot{v}(s) = \lim_{n\to\infty} 2^n (v_n(s) - s)$$

Notice that $\frac{ds}{dt}(t) = \dot{s}(t) = \dot{v}(s(t))$. Considering $\dot{v}(t)$ as a known function and $s(t)$ an indeterminate one, this identity becomes a differential equation

$$s' = \dot{v}(s)$$

Having reduced the problem to this form, we can now accomodate non-constant input functions. Specifically, instead of having $s'(t)$ be defined by a function $\dot{v}$ that depends only on $s(t)$, we allow it to depend on $i(t)$ as well. That is, we write

$$s' = u(s, i)$$

for some given function u. (Notice that i may now contain information about time, as well as be equal to time.)

Together with a function $a(s, i)$ that computes the output, we now have precisely the data needed to specify a behavior of a stateful system subject to the boundary condition given by the input function $i(t)$, $t \in [0, 1]$. This can be seen as the infinitesimal limit of the specification of Φ.

Remark. In order for the expression

$$\frac{1}{\epsilon} \cdot (s(t) - s(t + \epsilon))$$

to be meaningful, the space S must have linear structure on it. As it happens, the state spaces which usually appear in the dynamical systems of physics are in fact vector spaces. Thus, we see that the idea of having internal state realize a given continuous behavior naturally leads to the classical PDE view of dynamical systems.

8 Conclusion

We have seen that the notion of internal state arises naturally when one progresses from the concept of a function, which gives raw input–output relation, to that of a process, or a system, which evolves indefinitely as new input is provided.

In the discrete case, the extensional contribution of a state is captured by an infinite tree of its possible executions, which can be specified by a function from strings over inputs to actions. The length of the string provides how many clock ticks have elapsed since the execution has started.

In the continuous case, we are lead to the notion of dynamical systems as solutions of differential operators. As in the previous case, the crucial role is played by an intermediate structure, within which the intensional data of computation resides.

References

1. Tarski, A.: Introduction to logic and to the methodology of deductive sciences. Oxford university press (1941)
2. Gunter, C.A., Scott, D.S.: Semantic Domains. Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B), pp. 633–674 (1990)
3. Plasmeijer, R., van Eekelen, M.: Functional programming and parallel graph rewriting. Addison-wesley (1993)
4. Achten, P., Plasmeijer, R.: The Ins and Outs of Clean I/O. JFP 5(01), 81–110 (1995)
5. Lijnse, B.: TOP to the Rescue. PhD thesis (2013)
6. Plasmeijer, R., Lijnse, B., Michels, S., Achten, P., Koopman, P.: Task-Oriented Programming in a Pure Functional Language. In: Proceedings of the International Conference on Principles and Practice of Declarative Programming, PPDP 2012, Leuven, Belgium, pp. 195–206 (2012)