1999

Continuous Models of Computation for Logic Programs: Importing Continuous Mathematics into Logic Programming's Algorithmic Foundations

Howard A. Blair
Syracuse University, Department of Electrical Engineering and Computer Science, blair@top.cis.syr.edu

Fred Dushin
Syracuse University, Department of Electrical Engineering and Computer Science, fadushin@top.cis.syr.edu

David W. Jakel
Syracuse University, Department of Electrical Engineering and Computer Science, dwjaket@top.cis.syr.edu

Angel J. Rivera
Syracuse University, Department of Electrical Engineering and Computer Science, angel@top.cis.syr.edu

Metin Sezgin
Syracuse University, Department of Electrical Engineering and Computer Science, mtsezgin@top.cis.syr.edu

Continuous Models of Computation for Logic Programs: Importing Continuous Mathematics into Logic Programming's Algorithmic Foundations

Howard A. Blair*, Fred Dushin, David W. Jakel, Angel J. Rivera, and Metin Sezgin

Department of Electrical Engineering and Computer Science,
Syracuse University
Syracuse, New York 13244-4100 USA
{blair,fadushin,dwjakel,angel,mtsezgin}@top.cis.syr.edu

Abstract Logic programs may be construed as discrete-time and continuous-time dynamical systems with continuous states. Techniques for obtaining explicit formulations of such dynamical systems are presented and the computational performance of examples is presented. Extending 2-valued and n-valued logic to continuous-valued logic is shown to be unique, up to choosing the representations of the individual truth values as elements of a continuous field, provided that lowest degree polynomials are selected. In the case of 2-valued logic, the constraint that enables the uniqueness of the continualization is that the Jacobian matrices of the continualizations of the Boolean connectives have only affine entries. This property of the Jacobian matrix facilitates computation via gradient descent methods.

1 Orientation

This paper is lightly technical; proofs are omitted in favor of intuitive discussion. Some derivations of principal results are sketched.

The semantics of logic programs, in almost all of the versions that remain today as serious topics of research or as bases for implementation of logic programming systems, fundamentally rest on the notion of interpretation, or synonymously, *structure*, for a first-order language. Herbrand models are mere special cases, and far less special than usually supposed, since every structure for a first-order language L is elementarily equivalent to a quotient of an Herbrand model of a suitable extension of L, [BM98]. The semantics of intuitionistic logics, modal logics, and even multi-valued logics are engineered from this notion of structure. Many of these semantics have an associated operator analogous to the one-step consequence operator $\mathbf{T}_P$ originally studied by van Emden and Kowalski [vEK76]. Viewed through the associated operator (which partially incorporates the semantics of the program), a logic program P is a discrete-state, discrete-time dynamical system. If we move

* Address correspondence to this author.

to a continuous-valued logic, then there are analogues of these familiar semantic alternatives. We will discuss the move to continuous-valued logics, which we call *continualization*, below. Continuous-valued logic programs are *continuous-state*, discrete-time operators, when viewed through the various associated consequence operators that generalize those of the discrete case. The continualization methods that we discuss lead to not only continuous but also differentiable operators. From that property one can 1) associate continuous-state, *continuous-time* dynamical systems, 2) provide a model theory for them in continuous-valued logic, 3) embed classical logic into continuous-valued logic, and 4) obtain continuous-time dynamical systems *equivalent* to the programs one starts with in the sense that both systems have the same fixed points, i.e. models.

Various models of computation such as register machines, Turing machines, and the simple imperative programming language (*cf.* [Te94]) show that computation can be viewed in the following way: Imagine n natural number variables $X_1, \ldots, X_n$ initialized to an n-tuple of values. A relatively simple function iteratively updates these values until a relatively simple n-ary relation is satisfied. Then the answer is extracted, again in a simple way, such as projection out of the final n-tuple. More fundamentally, this point of view is embodied in Kleene's Normal Form Theorem [Sh67]. We can repeat what we just said this way: any computation takes a token on a lattice point in an n-space $\mathbf{R}^n$ and repeatedly jumps it around on lattice points until it lands in a desired region of the space that is easy to recognize, and an answer is projected out of the point where the token ends up. So, there arises the possibility of moving the token around smoothly as time progresses continuously. Call such a process a *smooth computation*. A continuous-time dynamical system is just a set of rules for moving a token around smoothly. Our point of view is that simple dynamical systems based on low degree polynomials play the same role in smooth computation as normal logic programs play in discrete computation.

It is important to dispose of two misconceptions. First, on reflection, one might suppose that the claims of the previous paragraph were really straightforward since in extending a discrete operator to a continuous operator there is so much freedom that just about anything can be crafted. This is *not* so under very simple linear algebraic restrictions. Continualization of a 2-valued or multi-valued logic involves choosing elements of a continuous field to stand for the underlying discrete truth-values. What freedom there is in continualization is highly structured. Subject to a simple constraint, a continualization is unique up to a dependence only on the elements of the field chosen to represent the discrete truth-values. Different choices of truth-value representations in the field produce linearly isomorphic continualizations. Thus, different choices of truth-value representations amount to a change of basis in the continualization.

For all of the preceding remarks, it remains that there is of course residual arbitrariness in continualization. Our response to this arbitrariness is to reflect on the distinction between the Euclidean plane and the set of all pairs of real numbers $\mathbf{R} \times \mathbf{R}$. An arbitrary choice of coordinates is involved in identifying the two spaces that is not determined by any natural geometric considerations [Bo86]. When one reflects on the consequences of Descartes' [De37] arbitrary imposition of coordinates on the Euclidean plane, the overwhelming response to the residual arbitrariness in Descartes' move is: so what?

The second misconception that the authors have frequently encountered, particularly among those well-versed in "engineering mathematics", is that dynamical systems are a special, rather limited case of systems of differential equations. This also is not so. This paper is not the place to elucidate this claim. The members of a nearly all-encompassing class of first and higher-order differential equations are representable as dynamical systems; we refer the interested reader to Hubbard & West, [HW95], and Guckenheimer & Holmes, [GH83].

Plan of the paper: We will begin by stating our contention about the fruitful relationship between logic programs and dynamical systems, and speculate on the long-range gains to be expected from pursuing investigations in that direction, and then we will relate our ideas to other existing work. The continualization of $2-$ and $n-$valued logic will then be presented with a discussion about the uniqueness of the continualization, and we will conclude the section on continualization with comments on the relationship of our continuous truth values to fuzzy logic. We will then present an example of a continuous-time dynamical system corresponding to a propositional program and show how the system tracks into the $\{0,1\}$-valued fixed points of the program. A second example involves a very small program which turns out to have remarkable discrete-time dynamics and from which every finite and infinite propositional program can be modularly built. We will conclude the paper with a program over a signature with seven propositional letters that produces the sudden emergence and sudden collapse of enormously complex limit cycles as the interpretation of its main connective in its clause bodies is continuously tuned. We exhibit this program primarily as an example of emergent phenomena in continuous-valued logic.

2 The Contention and a Caution

We are proposing a supplementary direction of research for logic programming to go in. There is no claim of exclusivity here. In fact, we believe it is essential to coordinate efforts in the direction we propose with all worthwhile programs of research in logic programming, and also nonmonotonic reasoning.

We contend that logic programs, equipped with any of a variety of semantics, are representable as discrete time continuous state dynamical systems. Datalog programs are subsequently translatable into finite dimensional continuous time dynamical systems as well. Translation of predicate programs with function symbols into continuous time dynamical systems is clearly possible, but here the mathematics is heavy going, involving linear operators on infinite dimensional vector spaces, if, as far as we can see at present, unnatural encodings are to be avoided. This is a beautiful and powerful area of exploration, but mostly premature at this time, we think, for logic programming purposes. Nevertheless, we mention in passing that the dynamical properties of logic programs acting on a Hilbert space is intriguing [Mu96].

Notions rooted in logic are usefully and naturally translatable into notions from continuous mathematics, and more importantly, from the perspective of computing with (formal) logic, we import for free the staggeringly enormous arsenal of mathematical technology available in connection with computing the trajectories of dynamical systems and understanding their structure.

The big question of course is how much of what is imported is actually useful for the purpose of computing with formal logic. We shall attempt to demonstrate in this paper that even the most elementary computational aspects of dynamical systems have immediate application for computing with formal logic. Computing (feasibly) with logic can be seen as effectively (and feasibly) finding models, which are, in turn, fixed points of a suitable operator. (The notion of models-as-fixed-points is a pivotal contribution of Reiter's which he exploited when introducing the notion of *extension* in default logic, [Re80].) Below, we will discuss an approach to computing supported and stable models of datalog programs using dynamical systems.

The desired fixed points associated with a particular program or theory are usefully seen as fixed points of an associated dynamical system. With regard to this latter perspective, one may get a glimmer of the possibilities by perusing the *Fix Point Theory on the Web* site.[1]

Our grand contention raises a vitally important caution. Suppose (perhaps at this stage it is just a supposition) that our contention has merit. There arises the temptation to go into dynamical systems and fixed point theory looking for applications that will drive logic programs. The area is vast, providing years of mathematical research to enter upon. The opportunities abound for writing a multitude of papers, each one amounting to show how an approach to fixed point finding may be suitably adapted to computationally driving logic programs in this or that class. The adaptability of gradient descent methods to finding models is probably a large enough area for investigation to start a small research industry that could go on for several years. *Most of this activity would probably be a catastrophic dilution of effort in logic programming.* So, we advocate a more cautious approach: whatever fruitful trails exist in dynamical systems and fixed point theory

[1] http://www.math.utep.edu/Faculty/khamsi/fixedpoint/fpt.html

for logic programming purposes, these trails are probably best explored by carefully judging their suitability for enhancing less exotic research programs in logic programming, and nonmonotonic reasoning. We mention in passing that *our own* primary concerns with regard to continualization techniques at present lie in (1) program and model complexity and (2) belief revision.

3 Long-term Expectations

In this section we will briefly indicate a number of lines of investigation that can be supported by continualization techniques, and argue for the likely payoff from each area of research that we mention. At this stage in our presentation it is necessary to mention that in section 6 we will present an example of a continuous-time dynamical system that computes the supported models of a ground datalog program via the most naive so-called *gradient descent* method, that of following continuous trajectories. (For a brief description of gradient descent and a picture illustrating trajectories, see section 6). The example is *not* intended to advocate that supported models be computed in such a manner. Rather, we illustrate the continuous-time approach in the way we do because it highlights in a simple way the idea of the *basin-of-attraction* of a fixed point along with *geometric intuition* for how the collection of these basins is structured. This idea is important in applying continualization in the areas of research that we indicate below. The purpose of the example also includes showing how continuous-time dynamical systems can be seen as conservative extensions, from a logical point of view, of ordinary logic programs based on 2-valued (as well as 3- or n-valued) logic. The range of techniques which may subsequently be brought to bear on the dynamical systems that arise is enormous.

Roughly, gradient-descent is concerned with finding global minima of functions. The problem may be thought of as finding the locations of least altitude on a surface which is the function's graph. One may hope to find such optima by guessing and then descending as rapidly as possible on the surface much as a skier may take the path of steepest descent to get to the bottom of the trail as rapidly as possible.

Transfer of neural net methods: Neural networks, whatever their shortcomings, still represent a significant success for artificial intelligence. Perceptrons, which can be seen as simple neural nets, were originally described by Minsky and Papert [MP69] in terms of gradient descent algorithms. Gradient descent methods play a huge role in training algorithms [WZ89], [MMR97], [AGPC90]. Logic programming enjoys an advantage over neural networks by being able to equip the object corresponding to the neural net, namely the program, with a declarative semantics. The lack of declarative semantics for neural nets is sometimes enthusiastically regarded as some mysterious virtue

they possess, having acquired abilities through training algorithms that could not have been feasibly programmed. If we think of a logic program as standing in place of a neural net, the expressive power available in logic programming permits concise formulation of much more robust formal systems than can be *readily* given with a neural net. It is not that neural nets cannot be used in this way, it is just that, as "linguistic" formalisms, logic programming constitutes a much higher level language than do neural network formalisms.

Neural nets enjoy an advantage over logic programs in that they robustly employ techniques from continuous mathematics for adaptation and operation, and also for their design. We are seeking the best of both worlds by trying to import similar techniques from continuous mathematics into the much higher level formalism offered by logic programs.

Transfer of fuzzy control methods: There is much controversy about the scientific quality of fuzzy logic as a development in *logic* [Pa91]. Still, fuzzy control, with its apparent basis in fuzzy logic, represents an important success for the area, and is having a greater impact on industrial-strength applications than is logic programming. It is not enough to dismiss this success as simply the result of the relative abundance of opportunities in different sorts of application/problem areas. The observations that we made above about the advantages of neural nets over logic programming also hold regarding the advantage fuzzy logic and control has over logic programming. Perusal of the fuzzy control literature (for example [KGK94,Ma77,TS85,TKK91]) seems to indicate that (1) fuzzy logic and control offers higher level formalisms that come closer to logic programming than do neural networks, and (2) the use of continuous mathematical techniques is somewhat, at least typically if not in principle, less robust than is seen in neural networks. There seems to be a trade-off: as the semantics of the formalism becomes increasingly clear and declarative, continuous mathematics drops away. Our contention is that this is neither desirable, nor intrinsically necessary, rather being due to accidents of developmental history and the concerns of the area's practitioners.

Equality relations in fuzzy control [KGK94] come close to logic programs as a relatively high level formalism. However, there is no high-level fuzzy formalism comparable to logic programming languages. Fuzzy control is more of a tool-box of techniques at the area's present state of development. Much stands to be gained, if the techniques and capabilities of fuzzy control techniques can be formally amalgamated into logic programming. Moreover, there has been some work in the foundations of fuzzy logic that is interesting from the point of view of mathematical logic. Here again, with careful selection, logic programming stands to gain at its foundations by incorporating this work from fuzzy logic [Ha98].

Comparative program behavior:. The fundamental problem that investigation into comparative program behavior treats is seen by asking, given two

programs P_1 and P_2, how are their supported models related? It is possible to smoothly deform P_1 into P_2. The well-known one-step consequence operator $\mathbf{T}_P$ enables a program P to be regarded as a transformation on interpretations of the program whose supported models are the transformations' fixed points. As P_1 is deformed along various paths to P_2, what happens to its fixed points? It turns out that in some cases the fixed points of P_2 are obtainable from the fixed points of P_1 by such deformations, and sometimes not. When failure occurs, the reasons involve singularities and basin boundaries, but systematic organized knowledge of what is going on remains to be developed. One payoff from such techniques is that it is possible to very rapidly find the supported models of a program at the other end of various deformations from P after all of the hard work and time spent on computing P's supported models.

One need not only be concerned with the motion of supported models as programs are smoothly altered. Take for example a deductive database with an intentional program, i.e. a program, to which a variety of collections of facts, i.e. extensional parts, can be adjoined. A user might ask whether a simple ground goal G follows from the intentional program together with its current collection of facts. A deduction of G (or a refutation of $\neg G$) becomes a smooth trajectory of a suitable representation of this deductive database's intentional part as a dynamical system. An extensional part, i.e. a collection of facts, together with a goal, becomes a starting point for a trajectory. As the starting point changes, how do the trajectories change? If we knew robust answers to that question, we would expect to obtain short-cuts to query processing in many cases.

Hybrid systems as constraint programs: A constraint program clause can be described as having the form

$$A \leftarrow \Gamma : \varphi$$

where $A \leftarrow \Gamma$ is simply a normal program clause and φ is a *constraint* which, loosely, is just something which takes on a truth value in possible models of the program. A *model* of a constraint program is then an interpretation M in which, after replacing each constraint φ by its truth value ν in M, the resulting program has M as a model. In continuous-valued logic one needs to know how a continuous truth-value contributed by a constraint is to be combined with the rest of the clause. It becomes possible to parameterize the propositional connectives in the clause bodies by parameters that depend upon ν. In the case of 2-valued normal logic programs, in which the negative literals can be seen as constraints, the outcome of constraint evaluation has the effect of varying discretely among a range of Horn clause programs, as is seen in the Gelfond-Lifschitz transform [GL88]. Thus the outcome of evaluating various constraints in a guessed model in continuous-valued logic can have the effect of *tuning* the program continuously among a continuum of programs. In the

final section of the paper we will see the how enormously complex behavior *emerges* from tuning continuous-valued programs.

One application is to a paradigmatic class of control problems. These control problems can be described as having two major components, classically described as the *plant* and the *controller*. The controller receives sensor data in the form of continuous values and has the problem of adjusting tunable parameters in the plant so as to cause the sensor values to stay within acceptable ranges. Thus the controller may be described as steering the plant. The controller has a theory, or *control-law* in order to perform the steering task. Every so often, say every 50 milliseconds, the plant process changes enough so that a new control-law has to be devised. The more important problem facing the controller is the latter task. Control-law revision is a discrete task. In effect, the controller has to revise a theory. However, in continuous-valued logic the revision might be accomplished by tuning the continuous-valued logical operations in the control-law using the outcome of constraint evaluation, where the constraint expresses a continuous-valued relationship among the sensor readings.

Belief revision and theory change: One approach to theory change involves the notion of contracting a theory so that it does not entail a particular proposition φ. Given a theory T, the contraction of T with respect to φ is a suitably chosen consistent subtheory T' of T that does not entail φ [AGM85]. The corresponding notion for dynamical systems can be developed as follows. A theory T can be identified with a class of models $\mathcal{M}$. A subtheory T' of T is then identifiable with a superclass of models $\mathcal{M}'$ of $\mathcal{M}$. The notion for dynamical systems corresponding to that of *model* is *fixed point*. Thus the contraction notion for dynamical systems corresponds to increasing the number of fixed points that avoid certain regions in the space of possible fixed points.

Random combinatorial search and complexity: The fixed points of a dynamical system have (possibly empty) *basins of attraction*. Given a combinatorial search problem such as to find a Hamiltonian circuit in a graph, we may represent the problem as one of finding a supported or a stable model of a logic program [MT98]; in turn as one of finding a fixed point of a dynamical system. Distinct solutions are represented as distinct models. If one guesses a solution then the dynamical system may or may not move the guess to an actual solution if any adjustment is necessary. If the guess is moved to an actual solution σ then the guess was in the basin of attraction of σ. Two questions arise: (1) Given a guess that is in the basin of attraction of a solution, how fast can the guess be adjusted to produce the actual solution? (2) How likely is the guess to be in the basin of attraction of some solution? The first question has to do with the *flow* of the system (this is a question about vector fields) and the second question has to do with the size of the basins.

If progress toward an actual solution has a rate bounded above zero, and trajectories have bounded arc-length, then producing a solution from a good guess is a linear-time process. The size of the union of all basins of attraction then becomes a representation of the probability of guessing correctly modulo a linear-time adjustment. Knowledge about how to build programs with large basins of attraction when represented as dynamical systems would provide knowledge of how to program combinatorial search problems with *probabilistically* low run-times.

The ability to tune the connectives in a program also permits the tuning of structural complexity. Take for example finite ground programs all of whose clauses have the form

$$A \leftarrow B \mid C$$

where A, B and C are atoms and $\mid$ is a Boolean connective. The problem of deciding whether a supported model exists in which not every atom is false is NP-complete when $\mid$ is taken as NAND (i.e not-both), and n^3 (where n is the number of clauses) when $\mid$ is taken as XOR (exclusive-or). So consider the following heuristic for deciding an instance of the former problem (where $\mid$ is NAND): replace NAND by XOR, find the solutions of the latter (XOR) problem, and drag the solutions over to possible solutions for the former (NAND) problem by subjecting $\mid$ to a deformation from XOR to NAND. We have only limited experience with the heuristic, but we do know that something interesting happens in cases when it fails. Namely, it appears from computational experiments that solutions obtained on the XOR side of the problem get crunched together in a singularity as they are subjected to a smooth deformation towards NAND. The singularity's location indicates where a sudden sharp discontinuity in the complexity of the associated satisfiability problem takes place.

4 Relationship to Prior Work

To the best of our knowledge there is no prior work that explores continuous-time dynamical systems *as such* as a model of computation that directly relates dynamical systems to logic programs, although clearly, much work in neural networks bears on dynamical systems as a model of computation.

Related work of a different character that bears on logic programs as continuous-time dynamical systems is due to Robert Paige, [Pa94]. Paige's work uses sophisticated techniques of finite differencing to translate from one programming language (just as a testbed example, SETL2) to another (again, just as an example, C). With dynamical systems (discrete or continuous time) one can continuously deform one logic program to another. Our own investigations have not yet examined any detailed relationship between our own work and Paige's, but that there is a relationship is clear and acquiring a detailed understanding deserves serious attention.

Much of the original inspiration for our work comes from the work of Melvin Fitting both on metric methods [Fi94] and on bi-lattice semantics [Fi91]. The metric methods work views a program from a particular class of logic programs as a certain kind of discrete-time dynamical system called a contractive iterated function system [Ba93] to obtain a unique supported model (i.e. fixed point) of the program. The bi-lattice semantics permits continuous-valued logic.

In [BDH97] two of us explored the relationship between logic programs and cellular automata [Wo86,Wo94,TM87]. Cellular automata are easily continualized into continuous-time dynamical systems. Moreover, if one is concerned not only with models of programs, but also with the computational paths (trajectories) of programs, Horn clause programs are sufficient for simulating *covered* (i.e. no local variables in clause bodies) normal logic programs. The result is made precise and is contained as a theorem in [BDH97]. Previously in [B-H96] we reported on how covered normal logic programs could be seen as elements of a metric space of bounded almost everywhere continuous functions.

Another major development is the optimization modeling language *Numerica* [VHDJ98,VHLD97,VH97]. The language allows users to solve hard nonlinear optimization problems expressed as ordinary differential equations (what we call here a *continuous-time dynamical system*) by sophisticated interval analysis. We see a strong natural affinity between *Numerica* and our own work. *Numerica* is an exciting tool and it behoove us, or anyone, to take full advantage of it in work regarding dynamical systems.

In comparison with *Numerica*, we are proposing to use dynamical systems to literally drive logic programs viewed dually as programs and as theories in continuous-valued logic. In fact, viewed in this manner, dynamical systems based on low-degree polynomials play exactly the same role for computing with respect to continuous-valued logic as normal predicate logic programs play with respect to 2- and 3-valued logic. In other words, *running* a dynamical system (by which we mean generating a trajectory) is to do a deduction in continuous-valued logic, literally. The emphasis in *Numerica* is to deduce, via numerical analysis methods (specifically involving interval analysis), the optima of a dynamical system. Our emphasis is not on the optima per se, but rather on the trajectories. In an example which we will develop in the next section, we will determine supported models of normal logic programs by gradient descent, which of course involves the search for global optima. But, the real point of the example is to illustrate how to obtain non-trivial continuous-time dynamical systems which are essentially fixed-point equivalent to the discrete-time one-step consequence operator associated with a given normal logic program. By "non-trivial" in this context, we mean a system that can be specified without knowing where the fixed points were located before we started.

The Quantitative Rule Sets (QRS) of van Emden [vE86] and paraconsistent programs, as formulated by Subrahmanian and Kifer, [KS92] are related to our work, since both approaches, particularly the former, and optionally the latter, use continuous-valued logic. The main distinction between our work and this earlier work involving continuous-valued logic is that finite propositional QRS programs and the paraconsistent programs are not literally everywhere differentiable, most particularly at the boundaries precisely where the special case of classical two-valued logics are obtained. Our current notion of program, that involves continuous-valued logic, is differentiable everywhere. Continuous-valued logic programs that lack this property block the final step to a continuous-time dynamical system. However, much of van Emden's results on QRS's carry over to our programs anyway.

Tucker, Tapia and Bennett [TTB85] introduced notions of differential and integral for Boolean functions from $\{0,1\}^n$ to $\{0,1\}$. Our continualizations have differentials that do not agree with theirs, except, coincidentally, for conjunction. However, the Tucker-Tapia-Bennett partial derivative of material implication with respect to the hypothesis is constantly 0, i.e. false. The intuition behind this outcome alludes us. In our case the same partial derivative is signed and turns out to be -1 times the logical negation of the conclusion, indicating that if the conclusion is true, no change takes place in the truth of the implication as the hypothesis changes, and if the conclusion is false, then the implication undergoes change in the direction opposite to the change in the hypothesis. This is arguably more natural. Still, our motivation is fundamentally computational, not semantical. We are not too much concerned with the meaning of our continuous truth values. As we will demonstrate below, the dynamical systems are intended for computational purposes, and can easily filter out non-$\{0,1\}$-valued fixed points. At the end of our discussion of continualization we will point out the relationship of our continualizations to fuzzy logic. It is a good fit by happenstance. So whatever value the reader may put on T-norms and T-co-norms, our conjunction and disjunctions satisfy the T-norm and co-norm postulates [KGK94]. Indeed, our continualizations of conjunction and disjunction are coincidentally the most commonly occurring continuous representations of these connectives in fuzzy logic, when we choose 0 to represent *false* and 1 to represent *true*.

5 Continualizing Propositional Connectives

The reader may wish to skip to the example below; however the proof of the uniqueness of continualization with respect to lowest degree choices of polynomials is remarkable for its brevity. Begin with n-valued logic where the semantics of the k-ary propositional connectives are given by various functions of the type

$$\{v_1, \ldots, v_n\}^k \longrightarrow \{v_1, \ldots, v_n\}$$

where $\{v_1, \ldots, v_n\}$ is the set of truth values. Let $\mathcal{B} = \{v_1, \ldots, v_n\}$. Choose your favorite field $\mathcal{F}$ (presumably a continuous field, but at this stage that isn't necessary) such that $\mathcal{B}$ can be embedded one-to-one into $\mathcal{F}$. Embed $\mathcal{B}$ into $\mathcal{F}$ as you like and regard $\mathcal{B}$ as a subset of $\mathcal{F}$. Consider linear combinations with respect to $\mathcal{F}$ of all functions of the type $\mathcal{B}^k \longrightarrow \mathcal{F}$ under pointwise addition and scalar multiplication. The set of all such functions is itself a vector space over $\mathcal{F}$ of dimension n^k. Call this vector space V. Now, consider any collection of functions from $\mathcal{F}^k$ to $\mathcal{F}$ that forms a vector space U, also of dimension n^k, such that the restriction of the functions in U from $\mathcal{F}^k$ to $\mathcal{B}^k$ is a mapping *onto* V. Then the restriction mapping is a linear isomorphism from U to V, from which it follows that there are no non-trivial automorphisms of U that respect the restriction mapping. That is, using the functions in U there is at most one way to generalize the functions in V.

We now choose U. The method involves a special case of a technique known as Lagrangian interpolation, with which the reader need not be familiar. Let U be the vector space of polynomials of degree $n-1$ in each of k variables with coefficients in $\mathcal{F}$ and variables in $x_1, \ldots, x_k$. For example, if $\mathcal{F}$ is the reals and $n = k = 2$, then U is the vector space of polynomials of the form

$$\lambda_1 + \lambda_2 x_1 + \lambda_3 x_2 + \lambda_4 x_1 x_2$$

where $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ are real numbers.

U is a vector space over $\mathcal{F}$ of dimension n^k. We have only to show that the restriction mapping is *onto* V. For this we have to show that for every function f in V there is a polynomial in U that agrees with f on $\{v_1, \ldots, v_n\}^k$. Define the n^k-many distinct polynomials

$$P_{i_1 \ldots i_k}(x_1, \ldots, x_k) = \Pi^n_{\substack{j=1 \\ j \neq i_1}} (x_1 - v_j) \cdots \Pi^n_{\substack{j=1 \\ j \neq i_k}} (x_k - v_j)$$

for each choice of $i_m = 1, \ldots, n$, $m = 1, \ldots, k$. Let

$$p_{i_1 \ldots i_k} = P_{i_1 \ldots i_k}(v_{i_1}, \ldots, v_{i_k}) .$$

From the way $P_{i_1 \ldots i_k}$ is defined, it follows that

$$p_{i_1 \ldots i_k} \neq 0 .$$

Then

$$p^{-1}_{i_1 \ldots i_k} P_{i_1 \ldots i_k}(v_{i_1}, \ldots, v_{i_k}) = 1$$

and

$$p^{-1}_{i_1 \ldots i_k} P_{i_1 \ldots i_k}(x_1, \ldots, x_k) = 0$$

whenever $(x_1, \ldots, x_k) \in \mathcal{B}^k - \{(v_{i_1}, \ldots, v_{i_k})\}$. It is in taking the reciprocal of $p_{i_1 \ldots i_k}$ that we needed a field, and not merely a ring.

Now, let $f \in V$ be given and let

$$Q_f(x_1, \ldots, x_k) = \Sigma^n_{i_1=1} \ldots \Sigma^n_{i_k=1} f(v_{i_1}, \ldots, v_{i_k}) p^{-1}_{i_1 \ldots i_k} P_{i_1 \ldots i_k}(x_1, \ldots, x_k)$$

Then,

$$Q_f(v_{i_1}, \ldots, v_{i_k}) = f(v_{i_1}, \ldots, v_{i_k})$$

for all $(v_{i_1}, \ldots, v_{i_k}) \in \{v_1, \ldots, v_n\}^k$. That is,

$$f = Q_f$$

on $\{v_1, \ldots, v_n\}^k$. So the restriction map from U to V is onto. Hence there is exactly one way to extend the functions of type

$$\{v_1, \ldots, v_n\}^k \longrightarrow \{v_1, \ldots, v_n\}$$

to polynomials over $\mathcal{F}$ of degree $n-1$ in each of k variables.

We call this construction the lowest degree polynomial *continualization* of n-valued logic over field $\mathcal{F}$, when $\mathcal{F}$ is the real or complex number field. We have just shown this continualization to be unique among possible lowest degree polynomials up to the representations in $\mathcal{F}$ of the truth values. Notice that changes in the choice of truth-value representations in the field amounts to a change of basis in U.

Notice also that if we continualized with not necessarily lowest degree polynomials, then the polynomials that continualize the standard basis of the vector space V are divisible by the polynomials in U that continualize those same standard basis elements, since these not necessarily lowest degree polynomials merely contain superfluous factors of the form $x_i - v$ where $v \in \{v_1, \ldots, v_n\}$. Thus the choice of the space of polynomials by which to continualize V is exceedingly limited by linear algebraic constraints.

Example: In the case of 2-valued logic, with truth values represented by 0 and 1, whether 0 corresponds to *false* or to *true* is immaterial to the continualization; instead, the correspondence of 0 with *false* or *true* determines the interpretation of the truth tables. For example, in the truth table below, suppose $b_1 = b_2 = b_3 = 0$ and $b_4 = 1$. Then the truth table defines conjunction if 0 represents *false*, and defines disjunction if 0 represents *true*.

Consider a function from $\{0,1\}^2$ to $\{0,1\}$ defined by the truth table

p	q	$p \bullet q$
0	0	b_1
0	1	b_2
1	0	b_3
1	1	b_4

Then

$$\begin{aligned} p \bullet q &= b_1(1-p)(1-q) + b_2(1-p)q + b_3 p(1-q) + b_4 pq \\ &= \lambda_1 + \lambda_2 p + \lambda_3 q + \lambda_4 pq \end{aligned}$$

where

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 1 & 0 & 0 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \end{bmatrix} .$$

Notice that by the uniqueness of the continualizing polynomial under the assumption that the exponents are bounded by 1, each of the Boolean sentence connectives, conjunction and disjunction, *must* be represented by one of the functions given by the two polynomials uv, $u+v-uv$, depending on whether 0 or 1 represents *false*. Also, in the case of unary Boolean functions, negation *must* be represented by the function given by $1-u$. The inverse of the above matrix gives the restriction mapping discussed in the derivation above, with respect to a fairly natural choice of bases.

The example shows that our lowest degree continualizations force on us the most typical choices for continualizing conjunction and disjunction in fuzzy logic. To the perhaps limited extent that one accepts the notion of fuzzy truth values, our truth values can be interpreted as having the same meaning, or lack of it, as fuzzy truth values have.

6 A Continuous-Time Example

In this section we will show by the development of an example how to represent a normal ground datalog program, i.e. a finite propositional normal logic program, as a continuous-time dynamical system. We shall not, in this paper, be rigorously detailed with these dynamical systems. The example is the following:

$$\begin{aligned} a &\leftarrow \neg b, \neg c \\ b &\leftarrow \neg a, \neg d \\ c &\leftarrow d \\ d &\leftarrow c \end{aligned}$$

With respect to the usual 2-valued semantics, this program has three supported Herbrand models, i.e. fixed points: $\{a\}$, $\{b\}$, $\{c,d\}$. The first two are stable, the third is not. Our methods have thus far concentrated on finding supported, (i.e. fixed point) models of programs. For propositional programs, a supported model can be checked for stability in linear time. Thus, regarding programs that do not have many more supported models than stable models, efficiently finding supported models is a useful heuristic for finding stable models.

A continuous-time dynamical system is a system of simultaneous equations of the form

$$\begin{aligned} \tfrac{dx_1}{dt} &= f_1(t, x_1, \ldots, x_n) \\ &\vdots \\ \tfrac{dx_n}{dt} &= f_n(t, x_1, \ldots, x_n) . \end{aligned}$$

Assumptions on the f_i vary according to the purpose at hand. In our applications the f_i are polynomials, and occasionally trigonometric, or exponential. Hence integrability and differentiability follow painlessly. Intuitively, t is time, and the equations express how a point is moving as a function of time and current position. If the f_i are all independent of t, then the system is said to be *autonomous.*

Let *false* be represented by 0, and *true* by 1. By the continualization method of the previous section we obtain the following transformation, whose $\{0,1\}$-valued fixed points are precisely the supported models of our program:

$$T\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 1-b-c+bc \\ 1-a-d+ad \\ d \\ c \end{bmatrix} .$$

Let $\mathbf{R}$ be the field of real numbers. As an operator on $\mathbf{R}^4$, T has fixed points along two curved lines that orthogonally intersect at the point $(a,b,c,d) = (\frac{1}{2},\frac{1}{2},0,0)$. Projected onto the a,b-plane, these lines are along the main diagonal $b=a$ and the orthogonal line $b=1-a$. The $\{0,1\}$-valued fixed points are $(1,0,0,0)$, $(0,1,0,0)$ and $(0,0,1,1)$ as we expect from the program. We shall see how the continuous-time dynamical system representation finds these fixed points.

Let I be the identity operator on $\mathbf{R}^4$ and put $R = T - I$. The kernel of R, i.e. the subset of $\mathbf{R}^4$ that R maps to $(0,0,0,0)$, is precisly the set of fixed points of T. The kernel of R can be found by gradient-descent methods, among many others. Of course, the kernel can be found analytically on this simple example, but gradient-descent methods are more easily scalable.

We show how to express the dynamical system corresponding to gradient descent for finding the kernel of R, and show how to augment it to filter out unwanted fixed points.

The *differential* of R denoted by dR is given by the Jacobian matrix of R. This matrix is the matrix of partial derivatives of the component functions of R. Specifically, if we express R by

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} 1-x_1-x_2-x_3+x_2x_3 \\ 1-x_1-x_2-x_4+x_1x_4 \\ x_4-x_3 \\ x_3-x_4 \end{bmatrix}$$

then

$$dR = \left[\frac{\partial y_i}{\partial x_j}\right]_{i=1,2,3,4}^{j=1,2,3,4}$$

(i indexes rows, j indexes columns) The partial derivatives all have the affine form

$$\frac{\partial y_i}{\partial x_j} = a_{ij} + b_{ij}x_{k_{ij}} .$$

in the case of 2-valued logic when we continualize by polynomials with degree 1 in each variable.

Specifically for our example,

$$dR = \begin{bmatrix} -1 & -1+x_3 & -1+x_2 & 0 \\ -1+x_4 & -1 & 0 & -1+x_1 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix}$$

A program can always be first normalized to an equivalent program in 2-valued logic that has at most two literals in the clause bodies. Programs in such a normal form will yield affine forms for the resulting partial derivatives. The differential is a linear function whose graph, in a suitably translated coordinate system, forms a hyperplane tangent to the graph of R.

The standard (Euclidean) norm of a point is its distance from the origin. We denote the norm of a point $\mathbf{x}$ in $\mathbf{R}^n$ by $||\mathbf{x}||$. Note that $T(\mathbf{x}) = \mathbf{x}$ iff $R(\mathbf{x}) = \mathbf{0}$ iff $||R(\mathbf{x})|| = 0$ iff $||R(\mathbf{x})||^2 = 0$. (Squaring maintains differentiability when the kernel of R is reached.) We will reach the kernel of R by moving on tangent hyperplanes to the graph of $||R||^2$ that produce the most rapid instantaneous decrease in $||R||^2$. The *gradient* of $||R||^2$ is a vector that lies in this tangent hyperplane in a direction corresponding to the most rapid instantaneous change in $||R||^2$ and is given by $(d||R||^2)^T$. It can be calculated that, as long as the Jacobian is defined for this square-norm function,

$$(d||R||^2)^T = 2(dR)^T R$$

(Superscript T denotes matrix transpose.) The matrices have been transposed here so that one obtains the differential of $||R||^2$ as a column vector.

An instantaneous movement of $\mathbf{x}$ is expressed by $d\mathbf{x}$. To obtain the fastest decrease in $||R||^2$ while remaining on its graph, one moves $\mathbf{x}$ according to

$$\frac{d\mathbf{x}}{dt} = -(d||R||^2)^T = -2(dR)^T R\,.$$

(When written out without matrix notation, the equations above will be seen to conform to our definition of autonomous continuous-time dynamical system.)

We would like the system to stop moving $\mathbf{x}$ exactly when $||R||^2$ reaches 0. In general, one expects a dynamical system like this to occasionally get stuck at so-called saddle points or local minima. By adding auxiliary variables to the system, one can prevent it from reaching fixed points except at global minima *because we already know in advance what the global minimum is*, although not *where* it is, namely when $||R||^2$ is 0. We will not go into detail on this last point. More interesting is how to prune out unwanted fixed points of the dynamical system. Consider

$$C(\mathbf{x}) = [(x_i(x_i - 1))]_{i=1,\ldots,n}$$

The associated dynamical system is given by

$$\frac{d\mathbf{x}}{dt} = -2(dC)^T C\,.$$

Almost all points moved by this dynamical system are attracted exclusively to the vertices of the unit n-cube. The only exception is the point at the center of the cube. The sum of one's original system with the $0,1$-attractor does the trick. (The reader should intuitively reflect that motion produced by the sum of differentials accumulates in general in a nonlinear way).

Finally, the dynamical system in which we are interested, is

$$\frac{d\mathbf{x}}{dt} = -2(dR)^T R - 2(dC)^T C\,.$$

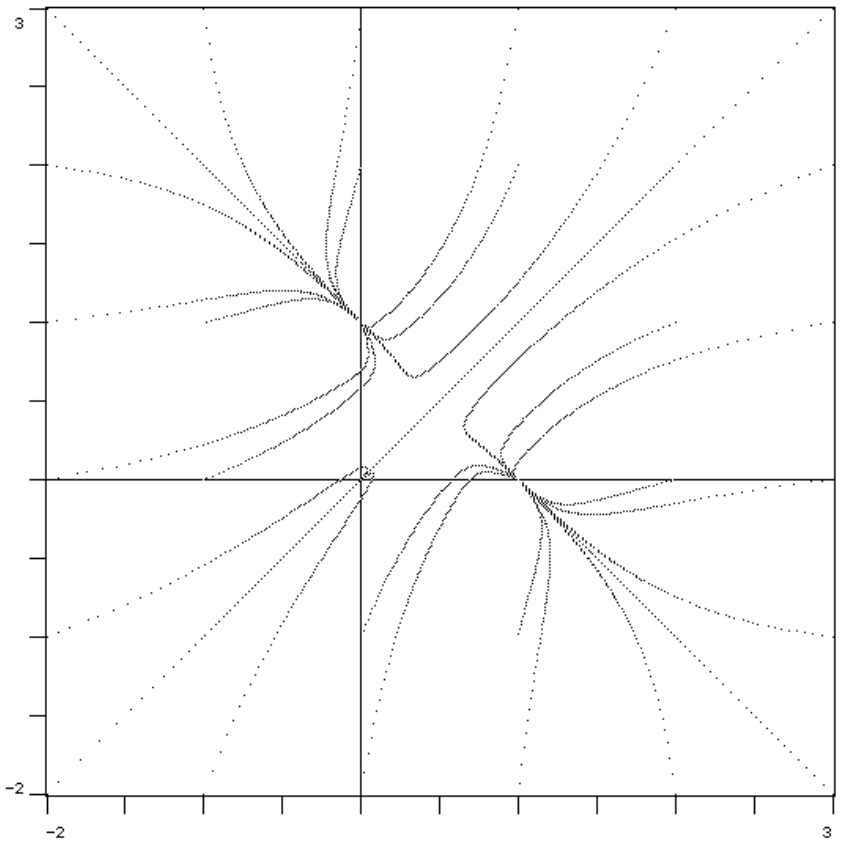
This system is almost fixed-point equivalent to the original logic program. It happens to have a saddle point, that is, a fixed point where further descent is still immediately possible. (We leave it to the reader to guess where.) Saddle points and local minima can be observed through auxiliary variables to correspond to values of $\mathbf{x}$ where the global minimum has not yet been achieved. The values of the auxiliary variables can thus be kept changing, destroying any possibility of achieving a fixed point of the dynamical system unless $\mathbf{x}$ is at a $\{0,1\}$-valued zero of $||R||^2$. These $\{0,1\}$-valued zeros of $||R||^2$ correspond to the supported models of the original program from which R was derived. For this reason, let us call a $\{0,1\}$-valued zero of $||R||^2$ a *supported model* of the dynamical system $\frac{d\mathbf{x}}{dt} = -2(dR)^T R - 2(dC)^T C$. We do not expect to reach one of these supported models no matter where we start; it is enough to be able to start with a reasonable probability of being in the basin of attraction of one of them. Repeated trials raise the probability as high as we like. Of course precise analysis of such probabilistic approaches is required, but at this stage we are offering only computational evidence of the utility of the approach. One approach to a complexity analysis involves estimating the volumes of the basins of attraction of the supported models.

Let $\mathbf{x}(t)$ be the position of a point at time t as it is determined by the system together with the starting position $\mathbf{x}(0)$. The position at time t can be calculated by iterating

$$\mathbf{x}(t + \Delta t) := \mathbf{x}(t) + \frac{d\mathbf{x}}{dt}\Delta t$$

where Δt approximates dt. In our example $\Delta t = 0.001$.

The figure below indicates the performance of the system in our example. The curves, called *trajectories*, that the dynamical system follows, begin at various integer lattice points in $\mathbf{R}^4$ and track into one of the supported models. The figure shows the trace of these trajectories on the (a,b)-plane.



7 A Fundamental Discrete-time System

Consider the program

$$a \leftarrow \neg b$$
$$b \leftarrow \neg a$$

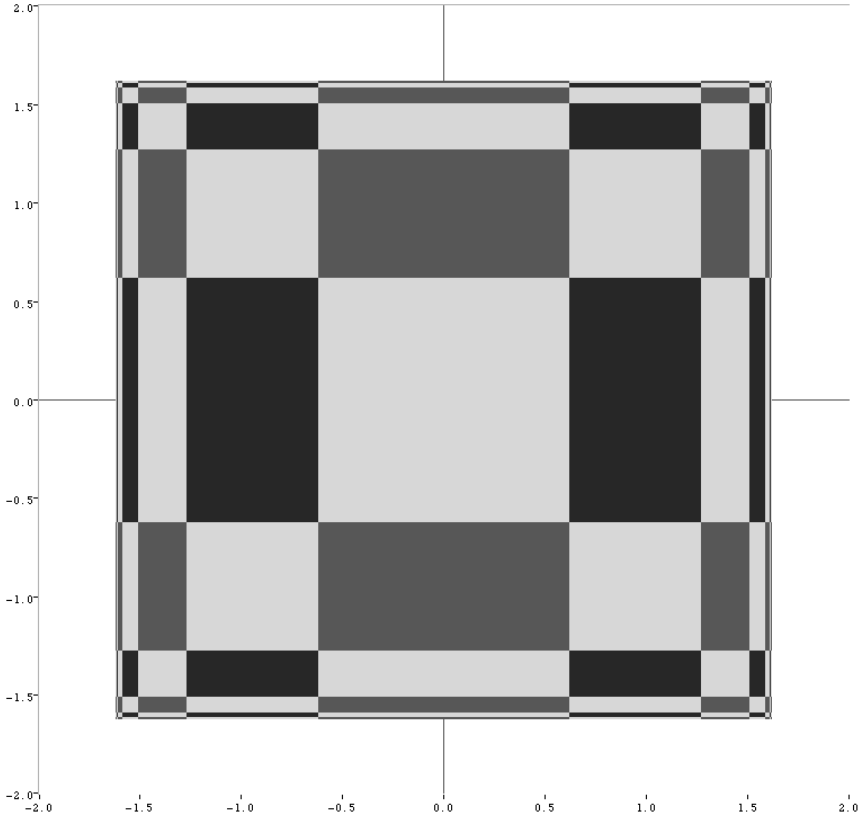
We take advantage of our 2-valued logic to re-express this program as the 2-valued equivalent

$$a \leftarrow b \,|\, b$$
$$b \leftarrow a \,|\, a$$

where the vertical bar indicates the NAND operation, i.e. not-both. As a discrete-time system with continuous states, we can express this program by

$$T \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 1 - b^2 \\ 1 - a^2 \end{bmatrix}$$

T has four fixed points: $(1,0)$, $(0,1)$, $(-\phi,-\phi)$, and $(-\hat{\phi},-\hat{\phi})$, where ϕ is the golden ratio, and $\hat{\phi}$ is $1-\phi$. Iterating T, we can think of this process as taking place in discrete time steps. The convergence of the dynamical system is expressed by the following figure.



The checkerboard pattern within the figure consists of countably many rows and columns, increasingly squeezed into the figure as the outer boundary is approached. The sequence of squares on the diagonals does not decrease in area in geometric proportion as the sequence approaches the corners of the checkerboard; rather, the decrease is determined by the dynamical (i.e. iterative) properties of a certain degree-4 polynomial on the interval from 0 to the golden ratio. Starting points in the plane outside the shaded checkerboard figure lead to diverging iterations to infinity. Inside the figure, starting points in the central small interior square and in all regions depicted by the same light shade of grey, are attracted to a limit cycle oscillating between $(0,0)$ and $(1,1)$. Starting points in any of the rectangular areas bearing the same shade of grey as the rectangle containing the point $(1,0)$ are attracted to $(1,0)$, and similarly for $(0,1)$. The remaining fixed points that involve the golden

ratio are repelling; all sequences of iterations that do not start on them are repelled away from them. This is also true of the entire boundaries between rectangular patches, as well as the outer boundary; any sequence starting on or near them is repelled away from them. The fixed point $(-\phi, -\phi)$ is at the lower corner of the checkerboard figure, and $(-\hat{\phi}, -\hat{\phi})$ is at the upper right corner of the central smaller lightly shaded square. Since the non-$\{0,1\}$-valued fixed points are repelling, they can be ignored.

Consider the following propositional program, which we call an if-then-else *component*.

$$\begin{array}{l} r \leftarrow u \mid v \\ u \leftarrow p \mid a \\ v \leftarrow q \mid b \\ a \leftarrow b \mid b \\ b \leftarrow a \mid a\,. \end{array}$$

The pattern in this program expresses

$$r \leftarrow (\text{if } a \text{ then } p \text{ else } q)$$

Suppose one has a program in which these clauses are a part. Any *non-repelling* fixed point of the program, even if it is not $\{0,1\}$-valued, will satisfy the if-then-else interpretation. From this it follows that one can represent any (including infinite ones) propositional program in 2-valued logic with a program built from repeated uses of the component. By the term *represent* we mean that given a program P, we can find a program Q built from if-then-else components whose non-repelling fixed points restrict to the fixed points of P. (Thus the completion of Q is a conservative extension of the completion of P in 2-valued logic.)

The attentive reader may have noticed that the programs we treated so far all have the property that heads of distinct clauses are distinct. More generally, heads of distinct clauses do not unify. We point out that it is always possible to represent a program by a conservative extension to a program with this property, even when the initial program is infinite.

8 Emergent Phenomena from Tuning

We conclude the paper with an example of a program that produces emergent phenomena in continuous-valued logic as its main connective is tuned.

$$\begin{array}{l} p_0 \leftarrow p_6 \bullet p_1 \\ p_1 \leftarrow p_0 \bullet p_2 \\ p_2 \leftarrow p_1 \bullet p_3 \\ p_3 \leftarrow p_2 \bullet p_4 \\ p_4 \leftarrow p_3 \bullet p_5 \\ p_5 \leftarrow p_4 \bullet p_6 \\ p_6 \leftarrow p_5 \bullet p_0 \end{array}$$

The possible interpretations under consideration for the $p \bullet q$ connective are given by

$$\lambda_1 + \lambda_2 p + \lambda_3 q + \lambda_4 pq \, .$$

with the λ_i as real numbers. The space of these polynomials is of course linearly isomorphic to $\mathbf{R}^4$. We will vary the coefficients $(\lambda_1, \lambda_2, \lambda_3, \lambda_4)$ of these polynomials, and hence the interpretation of the connective $\bullet$, along a short line segment in $\mathbf{R}^4$ between the points $(a_1, a_1 + b_1, a_1 - b_1, b_1)$ and $(a_2, a_2 + b_2, a_2 - b_2, b_2)$, where

$$\begin{aligned} a_1 &= -0.6892 \\ b_1 &= 0.3446 \\ a_2 &= -0.6911 \\ b_2 &= 0.3456 \end{aligned}$$

In the figure that follows, the horizontal λ-axis from 0 to 1 is the line segment in $\mathbf{R}^4$ from $(a_1, a_1+b_1, a_1-b_1, b_1)$ to $(a_2, a_2+b_2, a_2-b_2, b_2)$. Thus, each value on the horizontal axis corresponds to an interpretation of $\bullet$. The vertical η-axis corresponds to the Euclidean norm of valuations. We will briefly explain this by considering just one point plotted in the figure.

There is a point $(\lambda, \eta) = (0.375, 4.7051)$ occurring on one of the fibers in the portion of the figure just to the left of center. The interpretation of the connective $\bullet$ is given by the polynomial

$$a + (a + b)p + (a - b)q + bpq$$

where

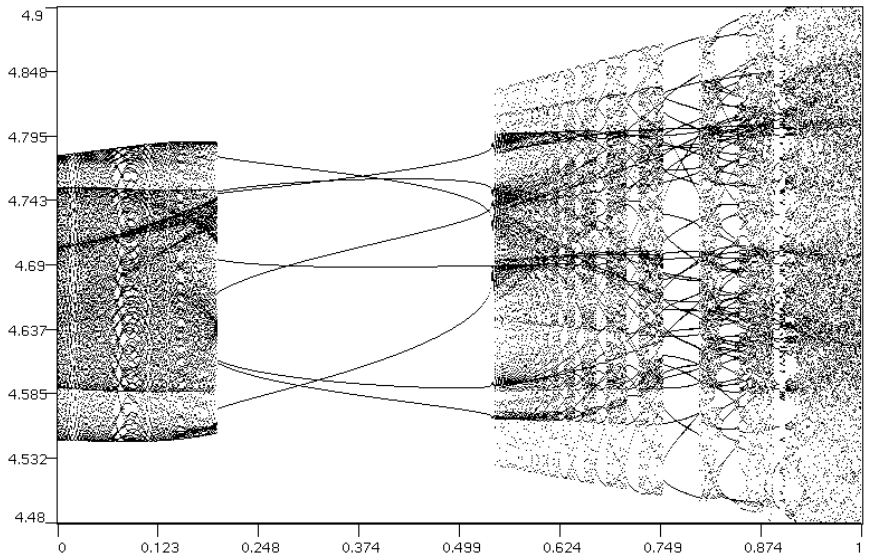
$$\begin{aligned} a &= (1 - \lambda)a_1 + \lambda a_2 \\ b &= (1 - \lambda)b_1 + \lambda b_2 \end{aligned}$$

where a_1, b_1, a_2 and b_2 are as above. Specifically, for the value $\lambda = 0.375$, the values of a and b are $a = -0.68991$ and $b = 0.34485$. The value $\eta = 4.7051$ is the Euclidean norm (the distance from the origin $(0,0,0,0,0,0,0)$) of the Herbrand interpretation (with continuous truth-values) given by $\mathbf{T}_P^{3197}(I_0)$ where $I_0(p_0) = 1$ and $I_0(p_1) \ldots = I_0(p_6) = 0$. That is, $I_0 = (1,0,0,0,0,0,0)$. (The reason we do not start the iteration at the origin of the space in this example is due to not wanting the truth value of every atom in the interpretations we reach to be equal). Specifically, the values of the atoms $p_0, \ldots p_6$ in $\mathbf{T}_P^{3197}(I_0)$ are given by

$$\begin{aligned} p_0 &= -1.1442045697971657 \\ p_1 &= -1.424646495185601 \\ p_2 &= -2.411334918922416 \\ p_3 &= -0.06419512110633097 \\ p_4 &= -2.3764341956984323 \\ p_5 &= -1.5553879344313004 \\ p_6 &= -2.2167841794959537 \, . \end{aligned}$$

What is the significance of the fact that this point (λ, η) occurs on this fiber? The fiber indicates an attracting limit cycle in the iteration of $\mathbf{T}_P$. The cycle has a period of 16. Only eight fibers are visible in the figure because the other eight are clustered in a similar figure corresponding to values with norms near 7 and are consequently off the top of the figure. Thus $\mathbf{T}_P$ is asymptotically converging to a limit cycle of period 16. Equivalently, the program corresponding to $\mathbf{T}_P^{16}$ has the valuation displayed above as a supported model. As the connective $\bullet$ is tuned continuously from left to right in the figure the model suddenly emerges out of somewhat chaotic iterative behavior at $\lambda = 0.197$, and appears to change continuously until it vanishes again at $\lambda = 0.542$. Actually, the structure of the fibers is collectively more complicated: there are three distinct values of λ where the eight fibers exchange places. This appears to be due to an irregular boundary of the basin of attraction around the fibers that captures the iteration of $\mathbf{T}_P$ at slightly different stages as the interpretation of $\bullet$ is tuned.

For each value on the horizontal axis, the program was iterated 2900 times beginning at $(1,0,0,0,0,0,0)$. If the norms of all of the valuations that result from an iteration are less then 64, then the iteration is deemed to not be diverging to infinity. A value on the vertical axis is the norm of a valuation. If the iteration is not diverging to infinity, then the norms of the next 300 iterations are plotted. The plots of every other iteration appear in the figure.



On the right-hand side of the figure the fibrous gaps are spreading out and becoming increasingly complex before somewhat abruptly disappearing as the cyclic structure in the iterations vanishes.

Although beyond the scope of this paper, we know how to find the regions of the space of polynomials that give rise to this emergent complex behavior. The fact that models can be made to vary continuously as the program is tuned appears to facilitate the application to hybrid control discussed earlier and similar applications.

Acknowledgements: We thank Krzysztof R. Apt and Victor W. Marek for valuable suggestions for improving the paper, and we give special thanks to Jeffrey B. Remmel for his time and effort in protracted technical discussion on several of the application topics discussed here. We also thank Lisa Ahn, of the Bronx High School of Science, for her participation in the seminar in which several of the ideas were developed. That the restriction mapping in section 5 is a linear isomorphism was pointed out to us by H.F. Mattson, Jr.

References

[AGM85] Alchourrón, C.E., Gärdenfors, P. & Makinson, D. "On the Logic of Theory Change", *J. Symbolic Logic*, 50(2):510–530, 1985.

[AGPC90] Anderson, J.A., Gately, M.T., P.A. Penz & Collins, D.R. "Radar Signal Categorization Using a Neural Network", *Proceedings of the IEEE*, 78:1646–1657, 1990.

[Ba93] Barnesly, M. *Fractals Everywhere*, Academic Press, 1993.

[B-H96] Blair, H.A., Chidella, J., Dushin, F., Ferry, A., Humenn, P. "A Continuum of Discrete Systems", *Annals of Mathematics and Artificial Intelligence*, vol. 21(2-4), 1997.

[BDH97] Blair. H.A., Dushin, F., Humenn, P. "Simulations Between Programs as Cellular Automata" in *Logic Programming and Nonmonotonic Reasoning*, Proceedings of the 4th International Conference, Dagstuhl, Germany, July, 1997. Jürgen Dix, Ulrich Furbach and Anil Nerode (eds.) Lecture Notes in Artificial Intelligence, 1997, pp. 115-131.

[BM98] Blair, H.A. & Marek, V.W. *Henkin Algebras with Applications to Default Logic*, presented at NMR'98, Trento, Italy.

[Bo86] Boothby, W.M. *An Introduction to Differentiable Manifolds and Riemannian Geometry*, Academic Press, 1986.

[De37] Descartes, R. *La Géométrie*, 1637.

[vE86] van Emden, M. H. "Quantitative Deduction and Its Fixpoint Theory", *Journal of Logic Programming*, vol. 3, no. 1, pp. 37-53. 1986.

[vEK76] van Emden, M. H. & Kowalski, R. A. "The Semantics of Logic as a Programming Language", JACM, vol 23, 1976. pp.733-742.

[Fi91] Fitting, M. "Bilattices and the Semantics of Logic Programming". JLP 11(1&2): 91-116 1991.

[Fi94] Fitting, M. "Metric methods, three examples and a theorem" *Journal of Logic Programming* volume 21, 1994, pp 113–127.

[GH83] Guckenheimer, J. & Holmes, P. *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields* Springer-Verlag Texts in Applied Mathematics, no. 42, 1983.

[GL88] Gelfond, M. & Lifschitz, V. "The Stable Model Semantics for Logic Programming." in *Logic Programming, Proceedings of the Fifth International Conference and Symposium*, Robert A. Kowalski, Kenneth A. Bowen (eds.), Seattle, Washington, pp. 1070–1080, 1988.

[Ha98] Hajek, P. *Metamathematics of Fuzzy Logic.* Kluwer, (to appear) 1998.

[HW95] Hubbard, J.H. & West, B.H. *Differential Equations: A Dynamical Systems Approach, Vols I & II.* Springer-Verlag Texts in Applied Mathematics, nos. 5 & 18, 1995.

[KGK94] Kruse, R., Gebhardt, J. & Klawonn, F. *Foundations of Fuzzy Systems*, Wiley, 1994.

[KS92] Kifer, M. & Subrahmanian, V.S. "Theory of Generalized Annotated Logic Programming and its Applications" JLP 12(3&4): 335-367 (1992)

[Ma77] Mamdani, E.H. "Application of Fuzzy Logic to Approximate Reasoning Using Linguistic Systems". *IEEE Transactions on Computers.* 26:1182-1191, 1977.

[MMR97] Mehrotra, K., Mohan, C.K. & Ranka, S. *Elements of Artificial Neural Networks*, MIT Press, 1997.

[MP69] Minsky, M. and Papert, S. *Perceptrons.* MIT Press, 1969.

[MT98] Marek, V.W. & Truszczyński, M. "Stable Logic Programming, an alternative logic programming paradigm." (This volume), 1998.

[Mu96] Mundici, D. *Private communication.*

[Pa91] Parikh, R. *Private communication.*

[Pa94] Paige, R. "Viewing a Program Transformation System at Work", *Programming Language Implementation and Logic Programming.* Proceedings: 6th International Symposium, PLILP'94, LNCS 844, Springer-Verlag, pp. 5–24, 1994.

[Re80] Reiter, R. "A Logic for Default Reasoning". *Artificial Intelligence*, 13:81–132, 1980.

[Sh67] Shoenfield, J. R. *Mathematical Logic.* Addison-Wesley, 1967.

[Te94] Tennent, R. D. "Denotational Semantics", in *Handbook of Logic in Computer Science, vol 3.* pp. 168–324. Oxford, 1994.

[TKK91] Takagi, H, Konda, T. & Kojima, Y. "Neural Networks Based on Approximate Reasoning Architecture". *Japanese Journal of Fuzzy Theory and Systems*, 3:63-74, 1991.

[TM87] Toffoli, Tommaso & Norman Margolis. *Cellular Automata Machines: a new environment for modeling.* MIT Press, 1987.

[TS85] Takagi, H. & Sugeno, M. "Fuzzy Identification of Systems and its Application to Modelling and Control". *IEEE Transactions on Systems, Man, and Cybernetics*, 15:116–132, 1985.

[TTB85] Tucker, T.H., Tapia, M.A. & Bennett, A.W., "Boolean Integral Calculus for Digital Systems", IEEE Transactions on Computers, c-34, no. 1, Jan. 1985, pp. 78–81.

[VH97] Van Hentenryck, P. "Numerica: A Modeling Language for Global Optimization", *IJCAI97.*

[VHDJ98] Van Hentenryck, P., Deville, Y. and Janssen, M. "Consistency Techniques in Ordinary Differential Equations", *Proceedings of the Fourth International Conference on Principles and Practice of Constraint Programming*, 1998.

[VHLD97] Van Hentenryck, P., Laurent, M. and Deville, Y. *Numerica: A Modeling Language for Global Optimization*, MITPRess, 1997.

[Wo86] Wolfram, Stephen. *Theory and Applications of Cellular Automata.* World Scientific, 1986.

[Wo94] Wolfram, Stephen. *Cellular Automata and Complexity.* Addison Wesley, 1994.

[WZ89] Williams, R.J. & Zipser, D. "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks", *Neural Computation*, 1:270-280, 1989.