12-1-1995

# Simulations Between Programs as Cellular Automata

Howard A. Blair
*Syracuse University, School of Computer and Information Science*, blair@top.cis.syr.edu

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

Polar Humenn
*Syracuse University*

SU-CIS-97-1

# *Simulations Between Programs as Cellular Automata*

Howard A. Blair, Fred Dushin, Polar Humenn

December 1995

*School of Computer and Information Science*
*Syracuse University*
*Suite 2-120, Center for Science and Technology*
*Syracuse, New York 13244-4100*

# Simulations Between Programs as Cellular Automata

**Howard A. Blair**[1] **Fred Dushin**
School of Computer and Information Science, Syracuse University
Syracuse, New York 13244-4100 USA
{blair,fadushin}@top.cis.syr.edu
fax:315.443.1122 tel:315.443.3565

**Polar Humenn**
BlackWatch Technology, Inc., 2-212 Case Center
Syracuse University
Syracuse, New York 13244-4100 USA
polar@blackwatch.com

**Keywords:** logic program, cellular automaton, orbit, simulation

## Abstract

We present cellular automata on appropriate digraphs and show that any covered normal logic program is a cellular automaton. Seeing programs as cellular automata shifts attention from classes of Herbrand models to *orbits* of Herbrand interpretations. Orbits capture both the declarative, model-theoretic meaning of programs as well as their inferential behavior. Logically and intentionally different programs can produce orbits that simulate each other. Simple examples of such behavior are compellingly exhibited with space-time diagrams of the programs as cellular automata. Construing a program as a cellular automaton leads to a general method for simulating any covered program with a Horn clause program.

## 1 Introduction

A covered [Su87] normal logic program is a cellular automaton (CA). The natural questions that arise in the context of cellular automata have to do with orbits of their initial states. Hence, by emphasizing the CA aspect of a logic program we our shifting attention from models of programs to *orbits* produced by programs.

Orbits provide a way to reveal dynamical relationships between various programs, but models remain important. Indeed, the portion of our work reported here has its origin in work of Melvin Fitting's on the subject of metric methods in the foundations of logic programming [Fi94]. Fitting discussed the way a set of Herbrand interpretations of some programs could be equipped with a complete metric such that the program becomes a contraction mapping on the set of interpretations. Hence by the Banach contraction mapping theorem, the program has a unique fixed point, i.e. a unique *supported* model. In terms of orbits, this means that all orbits converge to the program's unique supported model. We will give an example below that discusses *Fitting's program*: a program that Fitting discussed in his tutorial that is not a contraction mapping, has two supported models, and has orbits that chaotically swing back and forth between these supported models. The two supported models are the two accumulation points [Ke55] of the orbit, coming progressively closer to each one of the them for longer and longer periods with each swing.

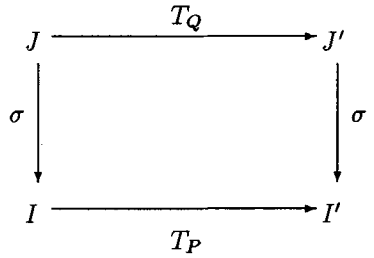
Orbits conveniently allow us to observe that in certain cases programs that are rather logically and intentionally inequivalent are nevertheless equivalent in a dynamical sense. The general idea is to set up a notion of the way Herbrand interpretations are simulated by other selected Herbrand interpretations. Then, for two programs $P$ and $Q$, and two Herbrand interpretations, $I$ for $P$ and $J$

[1] Address correspondence to this author.

for $Q$, one wants selected members of the orbit of $I$ with respect to $P$ to simulate selected members of the orbit of $J$ with respect to $Q$. The notion of simulation between Herbrand interpretations is to be formulated according to the purpose at hand, but should be compelling in each case.

A similar notion of simulation exists with the way Turing machines can be set up to simulate other Turing machines. The computations of Turing machines are orbits of initial instantaneous descriptions (i.e. tape-state configurations) [Ro67] of the machines. A universal Turing machine is universal in the sense that one can set up a notion of how selected instantaneous descriptions of the universal machine simulate instantaneous descriptions of any other machine. Given one machine $M$ and an instantaneous description $I$, one can find an instantaneous description of the universal machine whose orbit (i.e. computation) simulates the orbit of $I$ (i.e. computation proceding from $I$).

The situation for logic programs allows simulating orbits to be more tightly coupled to the simulated orbits than is the case just described above for Turing machines. In general one wants to set up a simulation of Herbrand interpretations $\sigma$ so that the following diagram commutes.




Call an Herbrand interpretation that assigns *false* to all but finitely many ground atoms *small.* Unfortunately the orbit of a small Herbrand interpretation in general does not stay among small interpretations. But we can often simulate programs with ones that produce orbits of small interpretations that remain small. This is the case with Fitting's program in example 2.2.

Informally, the concept of a *cellular automaton* (CA) begins with a space of interconnected *cells*, each being in a definite state at a definite time. The cells undergo state transitions. The range of possibilities for the states of a cell, in particular whether the set of cell states is discrete or continuous is an adjustable parameter in the concept. Important in the CA concept is that each cell's state transitions are determined by the cell's current state, together with the states of all cells to which it is *connected.* Call a cell $c$ together with all the cells to which it is connected the *neighborhood* of $c$. Neighborhoods are finite and we think of all the cells in a given neighborhood as being close together.

Conway's game of *Life* is a familiar example of a cellular automaton [Ga70, TM87]. There, the cell space consists, essentially, of the integer lattice points of the plane. Each cell, i.e. lattice point, is connected to the cells to the north, northeast, east, southeast, south, southwest, west, and northwest, as well as to itself.[2] The state of a cell is either on or off. The cells update their states synchronously according to the rule: A cell that is on (alive) remains on iff it is surrounded by 2 or 3 neighbors that are on, and a cell that is off (dead) turns on iff it is surrounded by exactly 3 live neighbors.

Covered logic programs are those in which every variable that occurs in the body of a program clause also occurs in that same clause's head. The ground call graph of such a program bears a resemblance to the situation with the cell space of Conway's life. A given ground atom occurs in the head of at most finitely many ground clauses and hence immediately depends on only finitely many ground atoms. Since there are only finitely many nonground clauses to begin with, there is a uniform finite bound on the number of atoms upon which a ground atom immediately depends. Moreover, if ground atom $A_1$ immediately depends on ground atom $A_2$, then every variable occurring in $A_2$ also occurs in $A_1$. $A_1$ and $A_2$ syntactically differ by a pair of nonground atoms present in the nonground

[2] The reader who is unfamiliar with *Life* is enthusiastically recommended to consult [Ga70] for a pleasurable introduction intended for general readers. Familiarity with Conway's Life is not required to read this paper, but will provide a general orientation to the subject of cellular automata.

program. Hence there is a uniform bound on how much "syntactic difference" there can be between two ground atoms, one of which immediately depends on the other; hence they are syntactically close.

We assume that the reader is familiar with the basics of the foundations of logic programming. An excellent, widely available introduction the subject is by K. R. Apt, [Ap90]. In most cases readers who are insufficiently familiar with logic programming will have their puzzlements cleared up with a few minutes' perusal of Apt's article.

Some notation: variables occurring in logic programs are denoted by $x$ and $y$ with or without subscripts, and constants are denoted $a$, $b$, and $c$ with or without subscripts.

## 2 Logic Programs as Cellular Automata

**Definition 2.1**: A *normal logic program* is a finite collection of clauses

$$A \leftarrow L_1 \wedge \cdots \wedge L_n$$

where each $L_i$ is a literal, and $A$ is an atom. Each $L_i$ and $A$ may be ground or nonground. $A$ is the *head*, and $L_1 \wedge \cdots \wedge L_n$ is the *body* of the clause. ∎

**Definition 2.2**: A clause is *covered* if every variable occurring in the body also occurs in the head. A program consisting of covered clauses is itself said to be *covered.* ∎

**Definition 2.3**: A formula is *ground* if it contains no free occurrences of variables. (Note that we regard normal clauses as open formulas, i.e., all occurring variables occur free.) If $P$ is a program, *ground*$(P)$ is the set of all ground instances of clauses in $P$. ∎

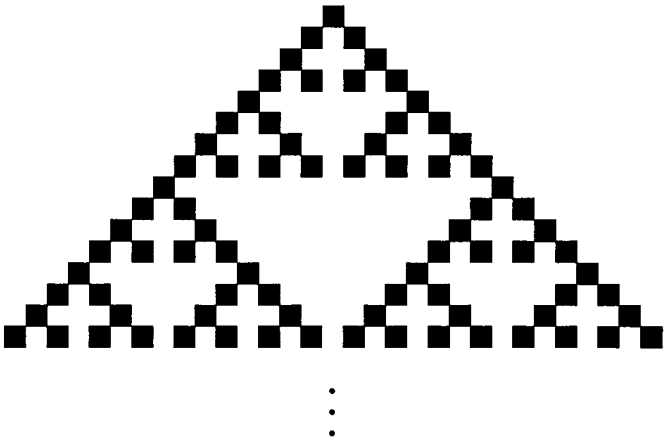
**Proposition 2.1**: If $P$ is a covered program and $A$ is a ground atom, then there are at most finitely many ground clauses in *ground*$(P)$ in which $A$ is the head. ∎

Covered programs constitute a computationally universal class of programs, as shown by Shepherdson [Sh91] and Nerode and Shore [NS93]. In particular, register machines are easily representable as covered Horn programs (i.e., all of the literals in the clause bodies are positive). The property that covered programs have that was given in the previous proposition permits us to view covered programs as cellular automata. Conversely, since left-bounded 1-dimensional cellular automata are easily representable as monadic covered programs, these programs also constitute a computationally universal class.

Formally, a cellular automaton (CA) consists of a *cell space* and a *state transition rule.* We take an ordered digraph with finite or countably many nodes, and an integer bound $k \geq 1$, such that the outdegree of every node is bounded by $k$ (outdegree$(c) \leq k$, for all nodes $c$.) If there is an edge from node $c$ to node $c'$ we say that $c$ is *connected* to $c'$. We assume that every node is directly connected to itself. The digraph is *ordered* in the sense that for each node $c$ the set of edges from $c$ is totally ordered. The digraph is the *cell space.* Each node is a *cell.* The CA has a finite set of *cell-states.* (The restriction that the set of cell-states be finite can be usefully relaxed, but we shall be considering CA's of this kind only in a subsequent paper). A *state* of the CA is a mapping from **cells**, the set of cells, to $S$, the set of cell-states.

For each cell $c$, let $\rho_c : S^r \rightarrow S$, where $r$ is the out-degree of cell $c$. Since $r \leq k$ and $S$ is finite, the set $\mathcal{B} = \{\rho_c \mid c \in \textbf{cells}\}$ is finite. Thus, $\rho$ is a function which maps a cell $c$ to its *update rule* $\rho_c$. Now, let $\mathcal{CS}$ be the set of states of the CA, let $\sigma$ be a state of the CA, and let $\sigma(c)$ be the state of cell $c$. The state transition rule is a mapping $\delta : \mathcal{CS} \rightarrow \mathcal{CS}$ defined by $(\delta(\sigma))(c) = \rho_c(\sigma(c_1), \ldots, \sigma(c_r))$ where $c_1, \cdots, c_r$ are the cells to which cell $c$ is connected in the order given.

**Example 2.1**: Consider the 1-dimensional CA in which each cell has a left and right neighbor. The state of each cell is either on or off, (*i.e.* 1 or 0) and each cell is updated simultaneously with every other cell by taking the exclusive-or (**xor**) of its left and right neighbors. In the figure that follows, each row is a sequence, indexed by the integers, of on/off states. The row immediately below a given row is obtained by applying the state transition rule. The first row contains exactly one cell which is on (cell number 0). The sequence of transitions procedes down the vertical axis of the figure.



The figure above is an example a *space-time diagram* of a 1-dimensional cellular automaton. The diagram depicts an *orbit* of the initial state of the cell space, i.e. a sequence of states of the cell space beginning with state depicted by the top row. In this example, the set of all cells is $\mathbf{Z}$, the integers. The set of edges can be given by $\{i \rightarrow j \mid |i-j| \leq 1\}$. Each cell $i$ has three outgoing edges $i \rightarrow (i-1)$, $i \rightarrow i$ and $i \rightarrow (i+1)$. $k = 3$, $S = \{0,1\}$, $\mathcal{B} = \{\beta\}$ where $\beta(p,q,r) = \mathtt{xor}(p,r)\}$, and $(\delta(\sigma))(i) = \sigma(i-1)$ **xor** $\sigma(i+1)$. ∎

**Theorem 2.1**: Every covered normal logic program is a CA.

**Proof:**

The set of cells of the cell space is the Herbrand base of the language $L$ of $P$ (i.e., the set of ground atoms of $L$.) The edges are determined as follows. Ground atom $A$ *refers to* ground atom $B$ with respect to $P$ if there exists a clause

$$A \leftarrow L_1 \wedge \cdots \wedge L_r$$

in *ground*$(P)$ such that $B$ is $L_i$ or $\neg B$ is $L_i$ for some $i = 1, \cdots, r$. For each pair of ground atoms $(A, B)$ such that $A$ refers to $B$ with respect to $P$, include edge $A \rightarrow B$. Let $n$ be the maximum number of literals occurring in any clause in $P$, and let $c$ be the number of clauses in $P$. Let $k$ be the maximum number of ground atoms that any ground atom $A$ is connected to via an outgoing edge. Since $P$ is covered, $k \leq cn$. The set of cell-states is $\{true, false\}$. A state of the automaton is an assignment of truth values to the ground atoms of $L$. Equivalently, a state of the automaton is an Herbrand interpretation for $L$.

Let $A$ be a ground atom. Suppose the clauses from *ground*$(P)$ in which $A$ occurs in the head are

$$A \leftarrow L_{11} \wedge \cdots \wedge L_{1n_1}$$
$$\vdots$$
$$A \leftarrow L_{m1} \wedge \cdots \wedge L_{mn_m}$$

Let $A_1, \ldots, A_{k_A}$ be the distinct ground atoms to which $A$ is connected (the order is actually immaterial, as in Conway's Life.) Let $I$ be an Herbrand interpretation for $L$ and define $\rho_A$ by

$$\rho_A(I(A_1), \ldots, I(A_{k_A})) = \bigvee_{i=1}^{m} \left( \bigwedge_{j=1}^{n_i} I(L_{ij}) \right)$$

Note that each atom occurring in the literals $L_{ij}$ is in the set $\{A_1, \ldots, A_{k_A}\}$. Each function $\rho_A$ is a Boolean function of arity at most $k$. Thus, there are at most $2^{2^k}$ such functions. ∎

We next give a precise definition of the term *orbit* for Herbrand interpretations with respect to programs and present an example of a simulation.

**Definition 2.4** Let $I$ be an Herbrand interpretation for $L(P)$. The *orbit* of $I$ with respect to program $P$ is the sequence

$$\langle T_P^n(I) \rangle_{n \in \omega}.$$

∎

We are referring throughout this paper to the program $P$ in the next example as Fitting's program whose origin was mentioned in the introduction. We must first make the notion of *supported model* precise.

**Definition 2.5**: Let $\mathcal{A}$ be a model (Herbrand or otherwise) of $P$. $\mathcal{A}$ is *supported* iff for each $\mathcal{A}$-valuation $v$ and clause head $A$ of $P$, if $\mathcal{A} \models_v A$, then for some clause body *Body* of $P$, $\mathcal{A} \models_v$ *Body*. ∎

It is an easy exercise to show that supported Herbrand models of $P$ are fixed points of the operator $T_P$ on Herbrand interpretations of $L(P)$.

**Example 2.2**: Let $P$ be

$$\begin{aligned} p(0) &\leftarrow \neg p(s(0)) \\ p(s(x)) &\leftarrow p(x), p(s(s(x))) \\ p(s(x)) &\leftarrow \neg p(x), \neg p(s(s(x))), \end{aligned}$$

and let $Q$ be

$$\begin{aligned} p(0) &\leftarrow \neg p(s(0)) \\ p(s(x)) &\leftarrow p(x), \neg p(s(s(x))) \\ p(s(x)) &\leftarrow \neg p(x), p(s(s(x))). \end{aligned}$$

Notice that longer and longer initial segments of both of the programs' supported models (fixed points of $\mathbf{T}_P$) are obtained as the orbit progresses. There is a natural sense in which $P$ and $Q$ simulate each other.

It is easy to calculate the two supported Herbrand models of $P$ and of $Q$:

$P$:

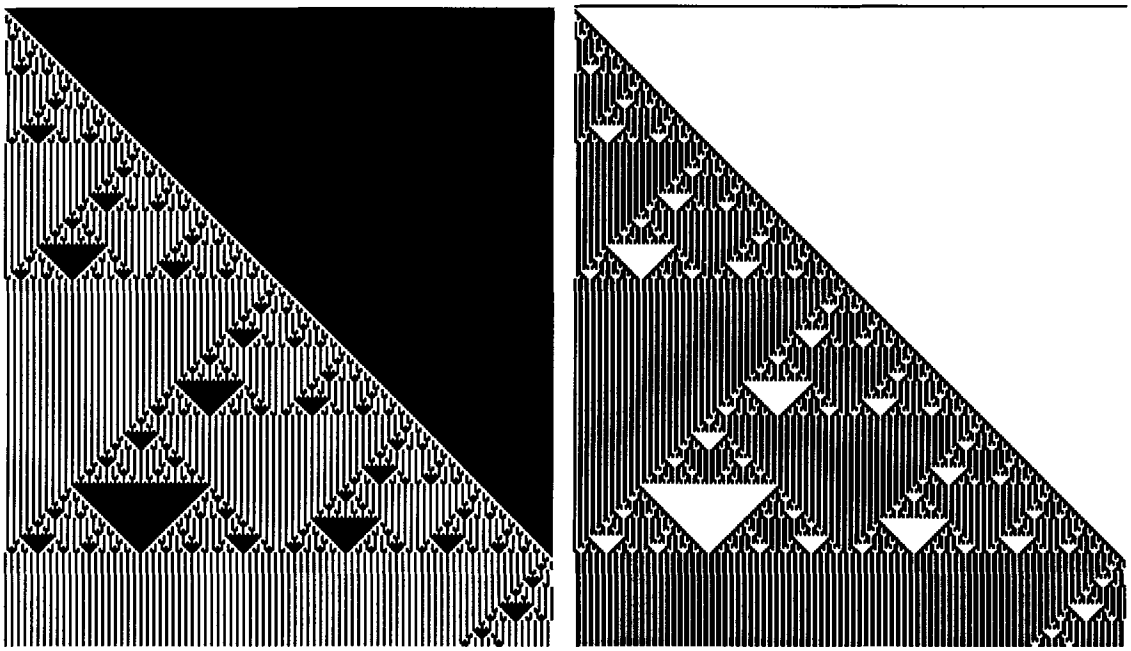
| $p(0)$ | $p(s(0))$ | $p(s(s(0)))$ | $p(s(s(s(0))))$ | $p(s(s(s(s(0)))))$ | ... |
|---|---|---|---|---|---|
| $F$ | $T$ | $F$ | $F$ | $T$ | ... |
| $T$ | $F$ | $F$ | $T$ | $F$ | ... |

$Q$:

| $p(0)$ | $p(s(0))$ | $p(s(s(0)))$ | $p(s(s(s(0))))$ | $p(s(s(s(s(0)))))$ | ... |
|---|---|---|---|---|---|
| $F$ | $T$ | $T$ | $F$ | $T$ | ... |
| $T$ | $F$ | $T$ | $T$ | $F$ | ... |

The simulation in this example is extremely simple, but we get wholesale returns for such a trifling investment. As mentioned in the introduction, the orbit of a small interpretation remains small. (Conway's Life has this property also.) The supported models of $P$ and $Q$ are seen to be complementary. If, for each Herbrand interpretation $I$ for a language $\mathcal{L}$ we define $\neg I(A) = \neg(I(A))$ for all

ground atoms $A$ of $\mathcal{L}$, then $\neg T_P(I) = T_Q(\neg I)$). Beginning with complementary Herbrand interpretations $I_0$ and $J_0$, where in $I_0$ all ground atoms are *false*, and in $J_0$ all ground atoms are *true*, we obtain the orbits displayed below. In the figures, the truth values of atoms $p(0)$ through $p(s^{232}(0))$ are displayed across each horizontal row; the $k^{th}$ row of the first figure is $I_k$ and the $k^{th}$ row of the second figure is $J_k$. $I_0$ and $J_0$ are at the top of each figure, respectively. Dark cells are assigned *true*. As we expect from the preceding remarks, $I_k$ and $J_k$ complement each other.



Clearly, for any pair of orbits of $P$ and $Q$, respectively that begin from complementary interpretations will be complementary throughout. Thus, relative to the notion that complementary Herbrand interpretations simulate each other, $P$ and $Q$ simulate each other. ∎

We will now give a uniform way to simulate a covered logic program with a definite clause program. The simulation is set up in two stages: First, given a program $P$, we first simulate $P$ by a general program $Q$ that we call the *case-simulation* of $P$. $Q$ has the property that each ground atom is an instance of at most one clause head.

Some notation: If $P$ is a program, the language determined by the constant function and predicate symbols of $P$ is denoted by $L(P)$.

The way in which an Herbrand interpretation of $L(P)$ is simulated by an Herbrand interpretation of $L(Q)$ has to do with *projection* and will become clear in the course of construction $Q$. Second, we construct a definite clause program $R$ to simulate $Q$. The difficulty regarding elimination of negations in the clauses of $Q$ centers on the ground atoms of $Q$ that are not instances of any clause head in $Q$. Given an orbit $\mathcal{O}$ of $Q$ with respect to an interpretation $I_0$, we want to be able to find an orbit $\mathcal{O}'$ of $R$ with respect to an interpretation $J_0$ such that $\mathcal{O}'$ simulates $\mathcal{O}$. We will be able to arrange for the interpretation $J_n$ of $\mathcal{O}'$ to simulate $I_n$ of $\mathcal{O}$, for every $n \in \omega$. The catch is that we get to pick $J_0$. Part of the criteria for selecting $J_0$ is that every ground atom of $L(P)$ that is not an instance of any clause head of $P$ will have a corresponding ground atom in $L(R)$ which is *false* in $J_0$. No Herbrand interpretation that assigns *true* to a ground atom which is not an instance of any clause head in a program $P$ can be a supported model of $P$. Hence, at least with respect to supported models such interpretations are irrelevant for $P$; that is, no supported model can be found outside this class of interpretations.

**Definition 2.6** An Herbrand interpretation $I$ is *relevant* for $P$ if every atom which is not an instance of any clause head of $P$ is *false* in $I$. ∎

We want positive simulations to treat orbits that begin from relevant Herbrand interpretations.

## 3 Simulations

We will show in this section that any covered normal logic program can be simulated, in a sense to be made precise, by a Horn clause program. Hereafter we use the more precise term *definite clause program* for Horn clause program. The construction of the simulation consists of two stages. The first obstacle is that different ground atoms with the same predicate symbol can be instances of different sets of clause heads in the given programs. To overcome this first obstacle one introduces a means of keeping track of which clause with a given predicate symbol occurring in its head yields a given ground atom. In PROLOG terminology we want only single-clause procedures. We call such programs *nonbacktracking*, and the nonbacktracking representation of a given program is called a *case-simulation*. The reduction to case-simulations is the first stage of the construction. The second stage uses new predicates to keep track of which ground atoms come out *false* after applying the familiar one-step consequence operator $\mathbf{T}_P$.

We begin by setting up case-simulations.

**Definition 3.1**: A *general program clause* is a formula of the form

$$A \leftarrow \varphi$$

where $A$ is an atom and $\varphi$ is a formula of first-order logic. A *general logic program* is a collection of general program clauses. We shall have need below of general logic programs in which either all of the clause bodies are formulas in conjunctive normal form (CNF) (conjunctions of disjunctions of literals), which we call *CNF-general logic programs*, or all of the clause bodies are formulas in disjunctive normal form (DNF) (disjunctions of conjunctions of literals), which we call *DNF-general* logic programs. ∎

In the definition that follows, the proliferating notation really makes the definition of *case-simulation* much more concise than it would otherwise be. We ask the reader to be patient; the example following the definition should clarify matters.

**Definition 3.2**: Let $q$ be a predicate symbol occurring in program $P$ and let

$$\begin{aligned} q(\mathbf{t}_1) &\leftarrow \varphi_1 \\ &\vdots \\ q(\mathbf{t}_n) &\leftarrow \varphi_n \end{aligned}$$

be all of the clauses in $P$ in which $q$ occurs in the head. We call each of these clauses a *case* of $q$. The *case-simulation* $R$ (which is a CNF-general program) of $P$ is obtained as follows. For each predicate symbol $q$ of $P$ replace the cases of $q$ by the clauses

$$\begin{aligned} q'(\mathbf{t}_1, c_1) &\leftarrow \varphi_1 \\ &\vdots \\ q'(\mathbf{t}_n, c_n) &\leftarrow \varphi_n \,. \end{aligned}$$

We assume that all of the predicate symbols $q'$ are new and distinct predicate symbols not previously occurring in $P$. Also, the constants $c_1, \ldots, c_n$, called *case-labels*, are assumed to be new and distinct. For each atom $A$, where $A$ is $q(\mathbf{t})$, $A^+$ is

$$q'(\mathbf{t}_1, c_1) \ \vee \cdots \vee \ q'(\mathbf{t}_n, c_n)$$

and $A^-$ is

$$\neg q'(\mathbf{t}_1, c_1) \wedge \cdots \wedge \neg q'(\mathbf{t}_n, c_n)$$

For a literal $L$ in the language of $P$, let $L^\circ$ be $A^+$ if $L$ is the positive literal $A$ and be $A^-$ if $L$ is the negative literal $\neg A$. Now, let $L_1 \wedge \cdots \wedge L_n$ be a conjunction of literals in the language of program $P$ and let $(L_1 \wedge \cdots \wedge L_n)^\circ$ is $L_1^\circ \wedge \cdots \wedge L_n^\circ$. $R$ is the result of replacing each clause body $\varphi$ in $P_q'$ by $\varphi^\circ$. ∎

**Definition 3.3** We define the *case-projection* of $A$ as the atom that results by replacing $q'$, the predicate symbol of $A$, by $q$ and deleting the last argument. Denote the case-projection of atom $A$ by $\pi(A)$, and let the case-projection of Herbrand interpretation $I$ be $\{\pi(A) \mid A \in I\}$, which we denote by $\pi(I)$. ∎

**Example 3.1** Let $P$ be Fitting's program. The case-simulation of $P$ is

$$\begin{aligned} p'(0,a) &\leftarrow \neg p'(s(0),a) \wedge \neg p'(s(0),b) \wedge \neg p'(S(0),c) \\ p'(s(x),b) &\leftarrow [p'(x,a) \vee p'(x,b) \vee p'(x,c)] \\ &\quad \wedge [p'(s(s(x)),a) \vee p'(s(s(x)),b) \vee p'(s(s(x)),c)] \\ p'(s(x),c) &\leftarrow \neg p'(x,a) \wedge \neg p'(x,b) \wedge \neg p'(x,c) \\ &\quad \wedge \neg p'(s(s(x)),a) \wedge \neg p'(s(s(x)),b) \wedge \neg p'(s(s(x)),c) \end{aligned}$$

The following table shows a portion of the the first seventeen interpretations of the orbit of $J_0$, the Herbrand interpretation for the case-simulation of $P$ where every ground atom is assigned *false*. The rows in bold-face give the result of projecting each interpretation $J_i$ to the language of $P$. The table illustrates the lemma that follows.

| | $J_0$ | $J_1$ | $J_2$ | $J_3$ | $J_4$ | $J_5$ | $J_6$ | $J_7$ | $J_8$ | $J_9$ | $J_{10}$ | $J_{11}$ | $J_{12}$ | $J_{13}$ | $J_{14}$ | $J_{15}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $p(0)$ | **F** | **T** | **F** | **F** | **T** | **T** | **T** | **F** | **F** | **F** | **F** | **T** | **F** | **F** | **F** | **F** |
| $p'(0,a)$ | F | T | F | F | T | T | T | T | F | F | F | T | F | F | F | F |
| $p'(0,b)$ | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F |
| $p'(0,c)$ | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F |
| $p(s(0))$ | **F** | **T** | **T** | **F** | **F** | **F** | **T** | **T** | **T** | **T** | **F** | **T** | **T** | **T** | **T** | **F** |
| $p'(s(0),a)$ | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F |
| $p'(s(0),b)$ | F | F | T | F | F | F | F | T | F | F | F | F | T | F | F | F |
| $p'(s(0),c)$ | F | T | F | F | F | F | F | F | T | T | F | T | F | T | T | F |
| $p(s(s(0)))$ | **F** | **T** | **T** | **T** | **F** | **F** | **T** | **F** | **F** | **T** | **F** | **T** | **F** | **F** | **T** | **T** |
| $p'(s(s(0)),a)$ | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F |
| $p'(s(s(0)),b)$ | F | F | T | T | F | F | F | F | F | T | F | F | F | F | T | T |
| $p'(s(s(0)),c)$ | F | T | F | F | F | F | T | F | F | F | F | T | F | F | F | F |
| $p(s(s(s(0))))$ | **F** | **T** | **T** | **T** | **T** | **F** | **F** | **F** | **T** | **F** | **F** | **F** | **F** | **T** | **T** | **T** |
| $p'(s(s(s(0))),a)$ | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F |
| $p'(s(s(s(0))),b)$ | F | F | T | T | T | F | F | F | F | F | F | F | F | F | F | T |
| $p'(s(s(s(0))),c)$ | F | T | F | F | F | F | F | F | T | F | F | F | F | T | T | F |
| $p(s(s(s(s(0)))))$ | **F** | **T** | **T** | **T** | **T** | **T** | **F** | **F** | **T** | **F** | **T** | **F** | **F** | **T** | **T** | **T** |
| $p'(s(s(s(s(0)))),a)$ | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F |
| $p'(s(s(s(s(0)))),b)$ | F | F | T | T | T | T | F | F | F | F | F | F | F | F | T | T |
| $p'(s(s(s(s(0)))),c)$ | F | T | F | F | F | F | F | F | T | F | F | F | F | F | F | F |

∎

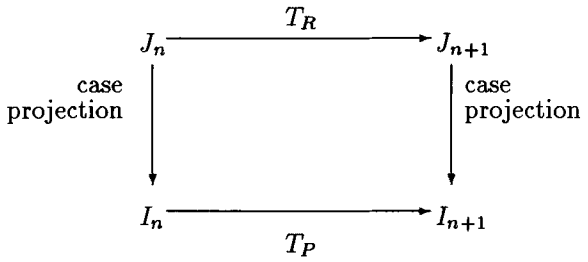
**Lemma 3.1** (Lemma on case-projections): Let $R$ be a case-simulation of $P$, let $I$ be an Herbrand interpretation of $L(P)$, and let $J$ be an Herbrand interpretation of $L(R)$ such that $I$ is a case-projection of $J$. Let $\langle I_n \rangle_{n \in \omega}$ be the orbit of $I$ with respect to $P$, and $\langle J_n \rangle_{n \in \omega}$ be the orbit of $J$ with respect to $R$. Then

$$I_n \text{ is a case-projection of } J_n, \text{ for all } n \in \omega\,.$$

**Proof:** Induction on $n$.
$n = 0$: By hypothesis, $I_0$ is a case-projection of $J_0$. For a ground atom $A$,

$A \in J_1\rceil_{L(P)}$
iff
$A \in T_P(J_0)$ and $A \in B_P$
iff (since $A \in L(P)$)
$q'(\mathbf{t}, c) \in J_0$, where $A$ is $q(\mathbf{t})$, for some case-constant $c$.
iff (since $I_0$ is a projection of $J_0$)
$A \in I_0$.

$n + 1$: Assume $I_n$ is a case-projection of $J_n$ and $J_{n+1}\rceil_{L(P)} = I_n$. The following diagram commutes:




To see this, we must show $I_{n+1}$ is a case-projection of $J_{n+1}$. For a ground atom $A$,

$A \in I_{n+1}$
iff
for some clause $A \leftarrow Body$ in $ground(P)$, $I_n \models Body$
iff (since $I_n$ is a case-projection of $J_n$)
for some clause $A' \leftarrow Body'$ in $ground(R)$ such that $\pi(A') = A$ and $\pi(Body') = Body$, $J_n \models Body'$.
iff
$A' \in J_{n+1}$, where $\pi(A') = A$.

Hence, $I_{n+1}$ is a case-projection of $J_{n+1}$. ∎

**Definition 3.4**: A logic program with the property that each ground atom is an instance of at most one clause head is called *nonbacktracking*. ∎

Case simulation programs are nonbacktracking.

Fix a first-order language $\mathcal{L}$. For each predicate symbol $p$ of $\mathcal{L}$, if $A$ is an atom $p(u_1, \ldots, u_k)$, let $\bar{A}$ denote the atom $\bar{p}(u_1, \ldots, u_k)$. If $L$ is a positive literal, let $L^*$ be $L$, and if $L$ is a negative literal $\neg A$, let $L^*$ be $\bar{A}$. If $\varphi$ is a formula in either DNF or CNF, let $\varphi^*$ be the result of replacing each occurrence of a literal $L$ by $L^*$.

We now set up the positive simulation of a nonbacktracking CNF-general program.

**Definition 3.5**: Let $P$ be a nonbacktracking CNF general program. The *positive simulation* $Q$ of $P$ is obtained as follows: We will first construct an intermediate program $Q_0$. For each predicate symbol $p$ of $L(P)$, introduce new predicate symbols $\bar{p}$ and $\hat{p}$ with the same polyadicity (same number of arguments as) $p$. For each non-unit clause $A \leftarrow \gamma$ of $P$, let $\beta$ be a formula in DNF equivalent to $\gamma$, and let $\psi$ be a formula in DNF equivalent to $\neg\gamma$, ($\gamma$ is in CNF), and include in $Q_0$ the clauses

$$A \leftarrow \beta^* \quad \text{and} \quad \overline{A} \leftarrow \psi^* .$$

No negative literals occur in the bodies of the clauses in $Q_0$, and each clause in $Q_0$ has the form

$$A_0 \leftarrow (A_{11} \wedge \cdots \wedge A_{1n_1}) \vee \cdots \vee (A_{m1} \wedge \cdots \wedge A_{mn_m}) .$$

For each such clause in $Q_0$ include in $Q$ the clauses

$$\begin{aligned} A_0 &\leftarrow A_{11} \wedge \cdots \wedge A_{1n_1} \\ &\vdots \\ A_0 &\leftarrow A_{m1} \wedge \cdots \wedge A_{mn_m} \end{aligned}$$

Lastly, include both in $Q_0$ and in $Q$ the clauses

$$\begin{aligned} \overline{p}(\mathbf{x}) &\leftarrow \hat{p}(\mathbf{x}) \\ \hat{p}(\mathbf{x}) &\leftarrow \hat{p}(\mathbf{x}) \end{aligned}$$

for each predicate symbol $p$ of $P$. $Q$ is a definite clause program. ∎

It is clear that any Herbrand interpretation $J$ of $Q$, is an Herbrand interpretation of $Q_0$, and $T_Q(J) = T_{Q_0}(J)$.

**Example 3.2**: Let $P$ be the case-simulation of Fitting's program. The positive simulation $Q$ of $P$ is obtained from the program $Q_0$ as in the definition. $Q_0$ is:

$$\begin{aligned} p'(0,a) &\leftarrow \overline{p}(s(0),a) \wedge \overline{p}(s(0),b) \wedge \overline{p}(s(0),c) \\ \overline{p}(0,a) &\leftarrow p'(s(0),a) \vee p'(s(0),b) \vee p'(s(0),c) \\ p'(s(x),b) &\leftarrow [p'(x,a) \wedge p'(s(s(x)),a)] \\ &\quad \vee [p'(x,a) \wedge p'(s(s(x)),b)] \\ &\quad \vee [p'(x,a) \wedge p'(s(s(x)),c)] \\ &\quad \vee [p'(x,b) \wedge p'(s(s(x)),a)] \\ &\quad \vee [p'(x,b) \wedge p'(s(s(x)),b)] \\ &\quad \vee [p'(x,b) \wedge p'(s(s(x)),c)] \\ &\quad \vee [p'(x,c) \wedge p'(s(s(x)),a)] \\ &\quad \vee [p'(x,c) \wedge p'(s(s(x)),b)] \\ &\quad \vee [p'(x,c) \wedge p'(s(s(x)),c)] \\ \overline{p}(s(x),b) &\leftarrow [\overline{p}(x,a) \wedge \overline{p}(x,b) \wedge \overline{p}(x,c)] \\ &\quad \vee [\overline{p}(s(s(x)),a) \wedge \overline{p}(s(s(x)),b) \wedge \overline{p}(s(s(x)),c)] \\ p'(s(x),c) &\leftarrow \overline{p}(x,a) \wedge \overline{p}(x,b) \wedge \overline{p}(x,c) \\ &\quad \wedge \overline{p}(s(s(x)),a) \wedge \overline{p}(s(s(x)),b) \wedge \overline{p}(s(s(x)),c) \\ \overline{p}(s(x),c) &\leftarrow p'(x,a) \vee p'(x,b) \vee p'(x,c) \\ &\quad \vee p'(s(s(x)),a) \vee p'(s(s(x)),b) \vee p'(s(s(x)),c) \\ \overline{p}(x) &\leftarrow \hat{p}(x) \\ \hat{p}(x) &\leftarrow \hat{p}(x) \end{aligned}$$

∎

The conversion of CNF clause bodies to DNF in the worst case results in generating exponentially many clauses in the positive simulation. However, this combinatorial explosion only occurs if one insists on presenting the positive simulation formally as a definite clause program. If one is content to let the clause bodies consist of formulas formed by conjunctions and disjunctions of atoms, then no combinatorial explosion results. Such programs are logically equivalent to definite clause programs, and produce the same orbits.

In order to have a simulation by a definite clause program we must set up the simulations between Herbrand interpretations.

**Definition 3.6** Let $P$ be a program and let $I$ be an Herbrand interpretation for $L(P)$, and let $L'$ be the expansion of $L$ to include new predicate symbols $\overline{p}$, $\hat{p}$ with the same polyadicity as $p$, for each predicate symbol $p$ of $L(P)$. An Herbrand interpretation $J$ for $L'$ is a *negation-simulation* of $I$ (with respect to $P$) iff the following conditions hold:

1. $I \models p(\mathbf{t})$ iff $J \models p(\mathbf{t})$
2. $I \models \neg p(\mathbf{t})$ iff $J \models \overline{p}(\mathbf{t})$
3. $J \models \hat{p}(\mathbf{t})$ iff $p(\mathbf{t})$ is not an instance of any clause head in $P$.

We shall write $M$ **NS** $N$ iff $M$ is the negation-simulation of $N$. ∎

Recall the definition of $\varphi^*$ for CNF and DNF formulas $\varphi$ used in the lemma on case-projections.

**Lemma 3.2** Let $\varphi$ be either a CNF or DNF ground formula and suppose $M$ **NS** $N$, where $N$ is an Herbrand interpretation for the language $\mathcal{L}$ in which $\varphi$ is a formula. Then

$$M \models \varphi^* \text{ iff } N \models \varphi.$$

∎

If $P$ is a program that is covered and nonbacktracking then each ground atom occurs in the head of at most one clause in *ground*$(P)$. This simple observation is crucial for the next theorem and explains why the construction of positive simulations has been confined to covered programs only.

**Theorem 3.1** Let $P$ be a covered nonbacktracking CNF-general logic program and let $Q$ be the positive simulation of $P$. Let $I$ be an Herbrand interpretation relevant for $P$. Let $J$ be the negation-simulation of $I$. Then

$$T_Q^n(J) \text{ is the negation-simulation of } T_P^n(I), \text{ for all } n \in \omega.$$

**Proof:**
Let $M$ be an Herbrand interpretation of $Q$ and $N$ an Herbrand interpretation of $P$.
$n = 0 : T_Q^0(J) = J$ **NS** $I = T_P^0(I)$.
$n + 1$ : Assume $J_n = T_Q^n(J)$ **NS** $T_P^n(I) = I_n$. We must show

$$p(\mathbf{t}) \in T_Q(J_n) \text{ iff } p(\mathbf{t}) \in T_P(I_n) \quad \text{and} \quad \overline{p}(\mathbf{t}) \in T_Q(J_n) \text{ iff } p(\mathbf{t}) \notin T_P(I_n)$$

for all ground atoms $p(\mathbf{t})$ of $L(P)$. Thus,

$p(\mathbf{t}) \in T_P(I_n)$
iff
for the unique clause $p(\mathbf{t}) \leftarrow \varphi$ in *ground*$(P)$ whose head is $p(\mathbf{t})$, $I_n \models \varphi$
iff
for the unique ground clause $p(\mathbf{t}) \leftarrow \varphi$ in *ground*$(P)$ whose head is $p(\mathbf{t})$, $I_n \models \beta$,
where $\beta$ is the formula in DNF equivalent to $\varphi$ such that $p(\mathbf{t}) \leftarrow \beta^*$
is the unique clause in *ground*$(Q_0)$ whose head is $p(\mathbf{t})$
iff (since $J_n$ **NS** $I_n$)
for the unique ground clause $p(\mathbf{t}) \leftarrow \varphi$ in *ground*$(P)$ whose head is $p(\mathbf{t})$, $J_n \models \beta^*$,
where $\beta$ is the formula in DNF equivalent to $\varphi$ such that $p(\mathbf{t}) \leftarrow \beta^*$
is the unique clause in *ground*$(Q_0)$ whose head is $p(\mathbf{t})$
iff
$p(\mathbf{t}) \in T_Q(J_n)$.

Also note that if $p(\mathbf{t})$ is not an instance of any clause head in $P$, then $p(\mathbf{t}) \notin T_Q(J_n)$.

Now, suppose $p(\mathbf{t})$ is an instance of some clause head in $P$. Then

$p(\mathbf{t}) \notin T_P(I_n)$
iff
for the unique clause $p(\mathbf{t}) \leftarrow \varphi$ in $ground(P)$ whose head is $p(\mathbf{t})$, $I_n \not\models \varphi$.
iff
for the unique clause $p(\mathbf{t}) \leftarrow \varphi$ in $ground(P)$ whose head is $p(\mathbf{t})$, $I_n \models \neg\varphi$
iff
for the unique clause $p(\mathbf{t}) \leftarrow \varphi$ in $ground(P)$ whose head is $p(\mathbf{t})$, $I_n \models \psi$, where $\psi$ is the formula in DNF equivalent to $\neg\varphi$ such that $\overline{p}(\mathbf{t}) \leftarrow \psi^*$ is the unique clause in $ground(Q_0)$ whose head is $\overline{p}(\mathbf{t})$.
iff
$\overline{p}(\mathbf{t}) \in T_{Q_0}(J_n)$. ∎

**Example 3.3**: The nonbacktracking program $P = \{(r(b),\ q(a),\ p(x) \leftarrow q(y)\}$ is not covered. The positive simulation that would result fails to simulate $P$ as may be seen by inspecting the interpreation $I = \{r(b), q(a), p(a)\}$. ∎

## 4 Concluding Remarks and Future Work

In prior work [B-H96] we studied analytical properties of covered monadic logic programs, programs such as in the example involving Fitting's program, and the relationship of such programs to finite automata and discrete dynamical systems. Although such programs constitute a severely restricted class with regard to expressive succinctness, the expressive power of the class is largely unknown. However, the class is computationally universal. The class is equivalent to what are called left-bounded 1-dimensional cellular automata. The latter class is computationally universal since Turing machines can be represented in it [Mi96]. The computational universality indicates that at least the full power of locally stratified programs [Pr88, BMS95] should be obtainable for covered monadic programs, but for the present, this remains a conjecture.

The relationship between the results reported in [B-H96] and the simulations discussed in the present paper is important. The monadic covered logic programs naturally fit into an infinite dimensional vector space with a rich Euclidean-like geometric structure, the space $\ell^2$ [B-H96, Ha82]. We conjecture that this result applies to all covered programs, not just the monadic ones, and this is the subject of an ongoing investigation. An Herbrand interpretation that assigns *false* to all but finitely many ground atoms also lives in (a tiny subspace of) $\ell^2$. Recall from the introduction that such interpretations are called *small*. The orbit of a small Herbrand interpretation in the space $\{0,1\}^\omega \subseteq \mathbf{R}^\omega$ escapes $\ell^2$ in general. But simulations can often be set up so that the simulating orbits that start small remain small. Why is this important?

Just as we want to know how models change as programs change, we want to know how orbits change. It should be clear that as a program is perturbed its models change in rather unmanageable ways: chaotically. So too for orbits. But these changes in the orbits are not completely arbitrary and unstructured. Considerations of how perturbations of programs induce changes in the orbits within $\mathbf{R}^\omega$ seems intractible; within $\ell^2$ the situation is much better due to the presence of a complete Pythagorean metric with respect to which convergence, divergence, contractions and attractors of orbits are being explored.

## References


[Ap90] Apt, K. R. "Logic Programming" in *Handbook of Theoretical Computer Science*, J. van Leeuwen, ed., Elsevier, 1990, pp. 494-574.

[B-H96] Blair, H.A., Jagan Chidella, Fred Dushin, Audrey Ferry & Polar Humenn. "A Continuum of Discrete Systems" *Annals of Mathematics and Artificial Intelligence*, to appear.

[BMS95] Blair, H.A., Marek, W. and Schlipf, J. "The Expressiveness of Locally Stratified Programs", *Annals of Mathematics and Artificial Intelligence*, 15(1995)209-229.

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

[Ga70] Gardner, Martin. "The Fantastic Combinations of John Conway's New Solitaire Game 'Life'," *Scientific American* 223(4) (April, 1970), pp. 120–123.

[Ha82] Halmos, P.R. *A Hilbert Space Problem Book* Springer-Verlag, Graduate Texts in Mathematics no. 19, 1982.

[Ke55] Kelly, J.L. *General Topology*, Van Nostrand, 1955, Reprinted by Springer-Verlag, Graduate Texts in Mathematics, no. 27.

[Mi96] Mitchell, Melanie (1996). *Computation in Cellular Automata: A Selected Review.* Santa Fe Institute Working Paper 96-09-074.

[NS93] Nerode, A. and R. Shore, *Logic for Applications*, Springer-Verlag, 1993.

[Pr88] Przymusinski, T. "On the Declarative Semantics of Deductive Databases and Logic Programs," in *Foundations of Deductive Databases and Logic Programming,* Jack Minker, ed. Morgan-Kaufmann, Los Altos, CA. 1988

[Ro67] Rogers, H. *Theory of Recursive Functions and Effective Computability.* McGraw-Hill, New York, 1967.

[Sh91] Shepherdson J. C. "Unsolvable Problems for SLDNF-Resolution", *Journal of Logic Programming*, 10(1), 1991, pp.19–22

[Su87] Subrahmanian, V.S. On the Semantics of Quantitative Logic Programs, *Proc. 4th IEEE Symp. on Logic Programming*, pps 173-182, Computer Society Press. Sept. 1987.

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