Logic Programming for Cellular Automata

Marcus Völker

RWTH Aachen University
Thomashofstraße 5, 52070 Aachen, Germany
(*e-mail:* marcus.voelker@rwth-aachen.de)

Katsumi Inoue

National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
(*e-mail:* inoue@nii.ac.jp)

submitted 29 April 2015; accepted 5 June 2015

Abstract

Cellular automata can represent real-world phenomena studied in physics and biology, and have been applied to intelligent systems like artificial life and multi-agent systems. In this paper, we study a semantic preserving transformation between cellular automata and normal logic programs based on the $T_{\mathcal{P}}$-operator. In particular, a subset of normal logic programs is shown to precisely correspond to the classic grid-based cellular automata such as Conway's Game of Life. We use two automaton models: one-way bounded cellular automata, for simplicity of construction, and unbounded cellular automata, which correspond to the classic definition of grid-based cellular automata. Using this construction, some computational theorems are easily proved regarding phenomena in the configurations of one-way bounded cellular automata, and some decidability results are newly obtained on the orbits of normal logic programs.

KEYWORDS: semantic foundations, $T_{\mathcal{P}}$-operator, supported models, cellular automata, decidability

1 Introduction

The study of dynamic systems has become more and more important recently, especially in the context of simulation and prediction of complex, physical or biological systems. One possible structure that is of importance for modeling dynamic systems is the *cellular automaton*, which has been studied extensively for half a century (Gardner 1970; Langton 1986; Wolfram 2002; Kari 2005). Cellular automata have also been applied to artificial life and multi-agent systems.

On the other hand, to analyze a system governed by the notion of state transition, logic programming can be a good tool to describe and model dynamic systems. As shown by (Inoue 2011), there is a correspondence between ground *normal logic programs* and *Boolean networks* (Kauffman 1993). This correspondence is realized by mapping configurations of a Boolean network to Herbrand interpretations of the equivalent logic program. The result is that calculating the next configuration of a

Boolean network is equivalent to using the $T_{\mathcal{P}}$-operator introduced by (Apt, Blair, and Walker 1988).

In this paper, we will introduce a lifted correspondence based on the fact that a cellular automaton can be viewed as a Boolean network with multiple cell states and an infinite amount of nodes. We will map configurations of a cellular automaton to interpretations of the corresponding logic program, and will show as well a translation to calculate one of these structures from the other. The cellular automata structure in this paper is defined as *grid-based cellular automata*, which include Conway's *Game of Life* (Gardner 1970). We will show a mapping from these automata to a subset of normal logic programs, using the $T_{\mathcal{P}}$-operator and the supported model semantics (Apt, Blair, and Walker 1988).

There are a few works that apply logic programming to cellular automata in the literature. A way to simulate a cellular automaton with a logic program has been brought up by (Syrjänen 2000). However, their approach is to simulate a cellular automaton *with* a logic program, so that some problems can be computed using Answer Set Programming, rather than to regard a cellular automaton *as* a logic program, which lends itself to different applications than our approach. A similar simulation has been considered by (Sakama and Inoue 2013), who model finite cellular automata and use the modelling to investigate the unpredictability of cellular automata such as the existence of a *Garden of Eden* for Game of Life.

Normal logic programs have been regarded as cellular automata by (Blair, Dushin, and Humenn 1997; Blair et al 1997). The main difference in our approaches lies in the structure of cellular automata we consider. While Blair *et al.* work with a very general automata model that can be used to capture a large class of logic programs (the class of *covered logic programs* (Subrahmanian 1987)), the cellular automata they consider are so general that reasoning about their properties becomes difficult. Having obtained a translation between cellular automata and logic programs, we can then apply several theorems from classical logic to prove theorems for cellular automata, about the existence of *still lives* and other phenomena in the configurations, for example. Especially, some (un)decidability results can be easily obtained for configurations of *one-way bounded cellular automata* as well as for *orbits* of logic programs.

2 Cellular Automata

A *cellular automaton* consists of the following components (Blair et al 1997): A *cell space*, which is a finite or countably infinite set of cells, a set of *states*, one of which is assigned to each cell of the cell space, and a *transition function*, which maps one configuration (which is an assignment of states to cells) of the cellular automaton to a successor configuration. We will define a cellular automaton with two properties in mind:

- Cells are arranged on an infinite Cartesian grid.
- The cells that determine a cell's successor state are in the neighborhood of the cell. This neighborhood is defined via a maximum distance with respect to some norm on the Cartesian space.

From these two properties, we define a cellular automaton as follows (Wolfram 1984):

Definition 1
A d-dimensional *unbounded cellular automaton* (UCA) is a quadruple $(\mathbb{Z}^d, Q, r, \rho)$, where

- $\mathbb{Z}^d$ is the cell space of the automaton, where $\mathbb{Z}$ is the set of integers.
- Q is the set of states a cell can have.
- r is the *radius* of a cell's neighborhood, which is a square of cells around the original cell. We can calculate the area of the neighborhood as $a := (2r+1)^d$. If $r = 1$ and $d = 2$, this neighborhood is equivalent to the *Moore neighborhood* (Kari 2005).
- $\rho : Q^a \to Q$ is a local transition function that maps the current states of a cell's neighbors to the next state of that cell.

Furthermore, we define a function $\sigma : \mathbb{Z}^d \to Q$ to be the *configuration* of the cellular automaton, assigning a state to each cell of the cell space. Finally, the transition function of the whole cellular automaton $\delta : (\mathbb{Z}^d \to Q) \to (\mathbb{Z}^d \to Q)$ is given by $(\delta(\sigma))(c) = \rho(\sigma(c_1), \ldots, \sigma(c_a))$, where $c_1, \ldots, c_a$ are the neighbors of the cell c. A consecutive sequence of configurations obtained by state transitions is called a *trajectory* (or an *orbit*) of the cellular automaton.

While this definition allows us to define well-known cellular automata such as Conway's Game of Life (Gardner 1970), it is not ideal for a translation from logic programs. Therefore, we will first consider the following modified definition in this paper:

Definition 2
A d-dimensional *one-way bounded cellular automaton* (OBCA) is a quadruple $(\mathbb{N}^d, Q, r, P)$, where

- $\mathbb{N}^d$ is the cell space of the automaton, where $\mathbb{N}$ is the set of natural numbers.
- Q is the set of states a cell can have.
- r is the radius of a cell's neighborhood. In contrast to an UCA, this neighborhood is not of the same size for all cells; cells that are close to the boundary of the cell space (i.e., at least one of the coordinates is smaller than the radius) have a smaller neighborhood. The area is given by $a_c := \Pi_{i=1}^{d}(y_i + r + 1), c = (x_1, \ldots, x_d), y_i = min(r, x_i)$.
- $P = \{\rho_k \mid k \in \{0, \ldots, r\}^d\}$ is a set of local transition functions. The transition function $\rho_{k(c)}$ assigned to a cell c is given by $k(c) = (min(r, c_1), \ldots, min(r, c_d))$. As a shorthand, we write the function assigned to a cell c as ρ_c. This function has a signature of $\rho_c : Q^{a_c} \to Q$

Again, we define a function $\sigma : \mathbb{N}^d \to Q$ to be the *configuration* of the cellular automaton and give the transition function by $(\delta(\sigma))(c) = \rho_c(\sigma(c_1), \ldots, \sigma(c_{a_c}))$.

3 Normal Logic Programs

We follow the definition of logic programs given in (Apt, Blair, and Walker 1988; Inoue 2011), which is based on the *supported model semantics*. A *normal logic program* (NLP) $\mathcal{P}$ is a set of rules of the form

$$A \leftarrow A_1 \wedge \cdots \wedge A_m \wedge \neg A_{m+1} \wedge \cdots \wedge \neg A_n$$

where A and A_i's are atoms ($n \geq m \geq 0$). For each rule R, we define that $h(R) = A$, $b^+(R) = \{A_1, \ldots, A_m\}$ and $b^-(R) = \{A_{m+1}, \ldots, A_n\}$. A set of ground atoms is a *model* of $\mathcal{P}$, if for any rule R and any substitution of variables θ, $b^+(R)\theta \subseteq I$ and $b^-(R)\theta \cap I = \emptyset$ imply that $h(R)\theta \in I$. A model I is a *supported model* if, for any $A \in I$, there exists a rule $R \in \mathcal{P}$ and a substitution θ such that (i) $A = h(R)\theta$, (ii) $b^+(R)\theta \subseteq I$, and (iii) $b^-(R)\theta \cap I = \emptyset$. Furthermore, we use the $T_\mathcal{P}$-operator given by (Apt, Blair, and Walker 1988):

$$T_\mathcal{P}(I) = \left\{ h(R)\theta \mid R \in \mathcal{P}, b^+(R)\theta \in I, b^-(R)\theta \cap I = \emptyset \right\}.$$

From this definition, it follows that I is a supported model iff $T_\mathcal{P}(I) = I$.

Given a logic program $\mathcal{P}$ and an Herbrand interpretation I, the *orbit* of I with respect to the $T_\mathcal{P}$ operator is the sequence $\langle {T_\mathcal{P}}^k(I) \rangle_{k \in \omega}$, where ${T_\mathcal{P}}^0(I) = I$ and ${T_\mathcal{P}}^{k+1}(I) = T_P({T_\mathcal{P}}^k(I))$ for $k = 0, 1, \ldots$ (Blair, Dushin, and Humenn 1997).

4 From Cellular Automata to Logic Programs

Before we will discuss the construction for a UCA, we will introduce the easier construction for OBCAs as a preliminary construction.

4.1 OBCA to NLP

Given an OBCA $\mathcal{A} = (\mathbb{N}^d, Q, r, P)$, we want to construct an NLP with the semantics that a configuration σ of the OBCA corresponds to an Herbrand interpretation I of the NLP. The configuration $\sigma' = \delta(\sigma)$ corresponds to the set $I' = T_\mathcal{P}(I)$. To this end, we construct a logic program with the following structure:

- A constant symbol 0.
- A unary function symbol $s/1$
- A set of predicates $\{p_i/d \mid 1 \leq i \leq \lceil log_2 |Q| \rceil\}$

The relation between the elements of the logic program's Herbrand base and the cell states of the automaton is as follows: A cell $c = (x_1, \ldots, x_d)$ is encoded as the tuple $(s^{x_1}(0), \ldots, s^{x_d}(0))$. ($s^x(0)$ denotes x applications of s to 0) This cell's state is encoded by the truth assignment of the predicates over this tuple in the given model. This encoding is done in a binary fashion, that is, we enumerate the states in Q as $Q = \{q_0, \ldots, q_{|Q|-1}\}$. We then choose predicates $\{p_{b_1}, \ldots, p_{b_m}\}$ such that

$$\sum_{i=1}^{m} 2^{b_i - 1} = t, \ \sigma(c) = q_t$$

For example, if in a two-dimensional automaton with 6 states the cell $c = (1, 2)$ has the state q_5, we encode this in I as $p_1(s(0), s(s(0))) \in I$, $p_3(s(0), s(s(0))) \in I$ and $p_2(s(0), s(s(0)) \notin I$, since $5 = 2^0 + 2^2$ implies that $p_1(c)$ and $p_3(c)$ are true while $p_2(c)$ is false.

We want to generate rules for the NLP from the local transition functions ρ_c according to the following algorithm. Let $k := \lceil log_2 |Q| \rceil$. We can use a binary encoding (see above) to encode each $q_i \in Q$ as an element of the Boolean product $\mathbb{B}^k$. Using this encoding, we generate a set of functions $\varphi_c : \mathbb{B}^{k \cdot a_c} \to \mathbb{B}^k$. We can then split the resulting vector of Booleans into its k components and create k separate functions $\varphi_{c,i} : \mathbb{B}^{k \cdot a_c} \to \mathbb{B}$. These functions are classical propositional formulas; therefore, we can convert them to disjunctive normal form, i.e.,

$$\varphi_{c,i}(x_1, \ldots, x_{k \cdot a_c}) = \bigvee_j \psi_{c,i,j}(x_1, \ldots, x_{k \cdot a_c})$$

where $\psi_{c,i,j}$ only contains (possibly negated) literals and the conjunction operator. These $\psi_{c,i,j}$ will form the basis for the logic program, and there will be exactly one rule for each $\psi_{c,i,j}$. In the following, $\nu(c, i)$ calculates the i-th cell of the cell c's neighborhood by some enumeration. This enumeration has to be equivalent to the one used for the ordering of the arguments of $\psi_{c,i,j}$. The rules have the following form:

$$\begin{aligned} p_i(\vartheta_{c,c,1}(X_1), \ldots, \vartheta_{c,c,d}(X_d)) \quad \leftarrow \quad & \psi_{c,i,j}(p_1(\vartheta_{c,\nu(c,1),1}(X_1), \ldots, \vartheta_{c,\nu(c,1),d}(X_d)), \\ & \ldots \\ & p_k(\vartheta_{c,\nu(c,1),1}(X_1), \ldots, \vartheta_{c,\nu(c,1),d}(X_d)), \\ & p_1(\vartheta_{c,\nu(c,2),1}(X_1), \ldots, \vartheta_{c,\nu(c,2),d}(X_d)), \\ & \ldots \\ & p_k(\vartheta_{c,\nu(c,a_c),1}(X_1), \ldots, \vartheta_{c,\nu(c,a_c),d}(X_d))) \end{aligned}$$

The functions ϑ ensure that the correct amount of applications of the successor function s are applied to the variables, or alternatively to the constant 0. They are defined as follows:

$$\vartheta_{c,c',d}(X) = \begin{cases} s^{c'_d}(0) & if\ c_d < r \\ s^{r + c'_d - c_d}(X) & otherwise \end{cases}$$

The next theorem can be proved in a similar way to (Inoue 2011, Theorem 3.2).

Theorem 1
Let $\mathcal{A}$ be an OBCA, and σ any configuration of $\mathcal{A}$. Let $\mathcal{P}_\mathcal{A}$ and I_σ be an NLP and an Herbrand interpretation obtained from $\mathcal{A}$ and σ by the above translation, respectively. Then, there is a one-to-one correspondence between the trajectory of $\mathcal{A}$ starting from σ and the orbit of I_σ with respect to $\mathcal{P}_\mathcal{A}$.

Example 1
Consider the one-dimensional OBCA $\mathcal{A} = (\mathbb{N}, \{0, 1, 2\}, 1, \{\rho_0, \rho_1\})$, where

$$\begin{aligned} \rho_0(c, r) &= r \\ \rho_1(l, c, r) &= l + r \pmod 3 \end{aligned}$$

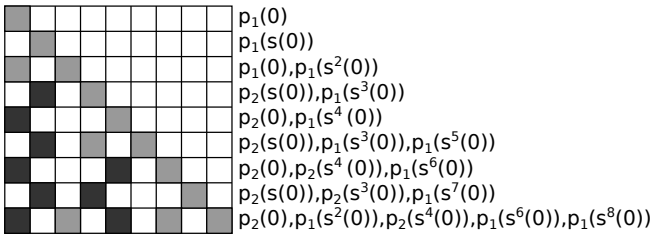

Fig. 1. Cellular automaton and corresponding Herbrand interpretations. For the states, white is 0, light grey is 1, and dark grey is 2

.

Splitting these functions into Boolean formulas yields:

$$\begin{aligned}
\varphi_{0,1}(c_1, c_2, r_1, r_2) &= r_1 \\
\varphi_{0,2}(c_1, c_2, r_1, r_2) &= r_2 \\
\varphi_{1,1}(l_1, l_2, c_1, c_2, r_1, r_2) &= (l_1 \wedge \neg r_1 \wedge \neg r_2) \vee (r_1 \wedge \neg l_1 \wedge \neg l_2) \vee (l_2 \wedge r_2) \\
\varphi_{1,2}(l_1, l_2, c_1, c_2, r_1, r_2) &= (l_2 \wedge \neg r_1 \wedge \neg r_2) \vee (r_2 \wedge \neg l_1 \wedge \neg l_2) \vee (l_1 \wedge r_1)
\end{aligned}$$

Further splitting the disjunctions yields the conjunctions:

$$\begin{aligned}
\psi_{0,1,0}(c_1, c_2, r_1, r_2) &= r_1 \\
\psi_{0,2,0}(c_1, c_2, r_1, r_2) &= r_2 \\
\psi_{1,1,0}(l_1, l_2, c_1, c_2, r_1, r_2) &= l_1 \wedge \neg r_1 \wedge \neg r_2 \\
\psi_{1,1,1}(l_1, l_2, c_1, c_2, r_1, r_2) &= r_1 \wedge \neg l_1 \wedge \neg l_2 \\
\psi_{1,1,2}(l_1, l_2, c_1, c_2, r_1, r_2) &= l_2 \wedge r_2 \\
\psi_{1,2,0}(l_1, l_2, c_1, c_2, r_1, r_2) &= l_2 \wedge \neg r_1 \wedge \neg r_2 \\
\psi_{1,2,1}(l_1, l_2, c_1, c_2, r_1, r_2) &= r_2 \wedge \neg l_1 \wedge \neg l_2 \\
\psi_{1,2,2}(l_1, l_2, c_1, c_2, r_1, r_2) &= l_1 \wedge r_1
\end{aligned}$$

These functions are converted into rules of the logic program, exemplarily done for $\psi_{0,1,0}$ and $\psi_{1,2,2}$:

$$\begin{aligned}
p_1(\vartheta_{0,0,1}(X_1)) \leftarrow \psi_{0,1,0}(\ & p_1(\vartheta_{0,0,1}(X_1)), p_2(\vartheta_{0,0,1}(X_1)), \\
& p_1(\vartheta_{0,1,1}(X_1)), p_2(\vartheta_{0,1,1}(X_1))) \\
p_1(\vartheta_{0,0,1}(X_1)) &\leftarrow p_1(\vartheta_{0,1,1}(X_1)) \\
\text{Hence, } p_1(0) &\leftarrow p_1(s(0)) \\
p_2(\vartheta_{1,1,1}(X_1)) &\leftarrow p_1(\vartheta_{1,0,1}(X_1)) \wedge p_1(\vartheta_{1,2,1}(X_1)) \\
\text{Hence, } p_2(s(X_1)) &\leftarrow p_1(X_1) \wedge p_1(s(s(X_1)))
\end{aligned}$$

The run of the CA on a configuration and the corresponding interpretations are depicted in Figure 1.

4.2 Extending the Construction for UCAs

To obtain the construction for a UCA $\mathcal{A} = (\mathbb{Z}^d, Q, r, \rho)$, we take the above construction and modify it to accomodate the bidirectionally infinite cellspace. One can view the cellspace $\mathbb{N}^d$ as one orthant of the d-dimensional space $\mathbb{Z}^d$. The basic idea of our construction is to view each orthant of $\mathbb{Z}^d$ as a seperate OBCA with the same, albeit mirrored behaviour. We accordingly generate similar rules for each orthant, and a set of rules for the hyperplanes at which at least one coordinate is in a radius around zero, i.e. where the orthants touch. There are multiple ways one can extend our previous construction to accomodate this cellspace. We chose the following idea: Instead of using one successor function s, we use two functions, s_+ and s_-. A cell $c = (x_1, \ldots, x_d)$ is now encoded as the tuple

$$(s^{|x_1|}_{\mathrm{sgn}(x_1)}(0), \ldots, s^{|x_d|}_{\mathrm{sgn}(x_d)}(0))$$

with sgn defined as:

$$\mathrm{sgn}(x) = \begin{cases} + & x \geq 0 \\ - & x < 0 \end{cases}$$

For example, the cell $(-2, 0, 3)$ is encoded as $(s^2_-(0), 0, s^3_+(0))$. The encoding of state using this cell encoding proceeds as with OBCAs, via a binary encoding using the predicates. To generate the rules, we reuse the concept from our OBCA construction. The key difference is that a UCA only has a single transition function ρ, but we have to generate multiple sets of rules depending on the orthant. To be specific, there need to be $(2r + 3)^d$ rulesets. For example, take a two-dimensional UCA with $r = 1$. This UCA needs 25 rulesets, with the following 5 terms for both coordinates: $s_-(s_-(X))$ for general negative values, $s_-(0)$ for -1, 0 for 0, $s_+(0)$ for 1, and $s_+(s_+(X))$ for general positive values.

Why we need to treat -1 and 1 seperately is easily shown with an example. Suppose we want a UCA in which every cell copies the cell to its left. If we just take $s_-(X)$, 0 and $s_+(X)$ as terms, we will generate a rule $p(s_+(X), Y) \leftarrow p(X, Y)$. However, this rule would also fire for $p(s_-(0), 0)$, leading to a configuration with $p(s_+(s_-(0)), 0)$ in its Herbrand interpretation. This is a forbidden atom, since it has no corresponding cell.

To prevent such inconsistencies, we have to ensure function symbols cannot be mixed, and this is achieved by adding these extra rules, so that the aforementioned automaton would instead generate the rules $p(s_+(s_+(X)), Y) \leftarrow p(s_+(X), Y)$ and $p(s_+(0), Y) \leftarrow p(0, Y)$.

To generate the rulesets, we use the same structure as in the OBCA construction, but with the following adapted function ϑ:

$$\vartheta_{c,c',d}(X) = \begin{cases} s^{|c'_d|}_{\mathrm{sgn}(c'_d)}(0) & if\ |c_d| \leq r \\ s^{|c'_d|}_{\mathrm{sgn}(c'_d)}(X) & otherwise \end{cases}$$

Example 2

Consider the one-dimensional OBCA $\mathcal{A} = (\mathbb{Z}, \{0, 1\}, 1, \rho)$, where $\rho(l, c, r) = \min(l, r)$

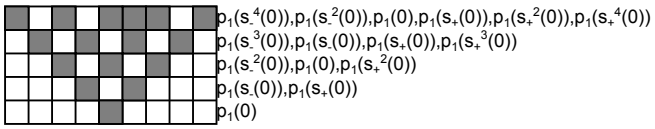

Fig. 2. Cellular automaton and corresponding Herbrand interpretations. For the states, white is 0, and grey is 1

.

Splitting this function into Boolean formulas yields:

$$\varphi_1(l_1, c_1, r_1) = \psi_{1,0}(l_1, c_1, r_1) = l_1 \wedge r_1$$

This function is converted into rules of the logic program:

$$\begin{aligned}
& p_1(\vartheta_{-2,-2,1}(X_1)) \leftarrow p_1(\vartheta_{-2,-3,1}(X_1)) \wedge p_1(\vartheta_{-2,-1,1}(X_1)) \\
\text{Hence, } & p_1(s_-(s_-(X_1))) \leftarrow p_1(s_-(s_-(s_-(X_1)))) \wedge p_1(s_-(X_1)) \\
& p_1(\vartheta_{-1,-1,1}(X_1)) \leftarrow p_1(\vartheta_{-1,-2,1}(X_1)) \wedge p_1(\vartheta_{-1,0,1}(X_1)) \\
\text{Hence, } & p_1(s_-(0)) \leftarrow p_1(s_-(s_-(0))) \wedge p_1(0) \\
& p_1(\vartheta_{0,0,1}(X_1)) \leftarrow p_1(\vartheta_{0,-1,1}(X_1)) \wedge p_1(\vartheta_{0,1,1}(X_1)) \\
\text{Hence, } & p_1(0) \leftarrow p_1(s_-(0)) \wedge p_1(s_+(0)) \\
& p_1(\vartheta_{1,1,1}(X_1)) \leftarrow p_1(\vartheta_{1,0,1}(X_1)) \wedge p_1(\vartheta_{1,2,1}(X_1)) \\
\text{Hence, } & p_1(s_+(0)) \leftarrow p_1(0) \wedge p_1(s_+(s_+(0))) \\
& p_1(\vartheta_{2,2,1}(X_1)) \leftarrow p_1(\vartheta_{2,1,1}(X_1)) \wedge p_1(\vartheta_{2,3,1}(X_1)) \\
\text{Hence, } & p_1(s_+(s_+(X_1))) \leftarrow p_1(s_+(X_1)) \wedge p_1(s_+(s_+(s_+(X_1))))
\end{aligned}$$

The run of the CA on a configuration and the corresponding interpretations are depicted in Figure 2.

4.3 A Note on the Inverse Construction

As we have demonstrated, there are constructions for both OBCAs and UCAs that give an equivalent NLP. The converse, however, is not generally true. It is trivially easy to construct an NLP that has no corresponding cellular automaton, because it violates locality.

Example 3
Consider the NLP

$$p(0) \leftarrow p(X)$$

Taken as a one-dimensional OBCA, the cell 0 is influenced by an infinite number of cells, which cannot be represented as a finite neighborhood in a grid.

To obtain a set of NLPs for which we can find a corresponding cellular automaton, we have to place strong restrictions on the form of the NLPs. By doing this, we essentially obtain the set of NLPs that are generated by our construction from cellular automata, plus transformations that do not change the meaning of the NLP.

Essentially, the NLPs we can translate back to OBCAs have to use only predicates of the same arity, the terms in the heads can be only 0 to $s^i(0)$ and $s^{i+1}(X_j)$, and the terms in the body have to be the terms in the head at the same position, with the only difference being the application depth of s.

Example 4
This is an LP that can be translated back to a CA

$$p(s^2(X_1), s(0), 0) \leftarrow p_1(s^3(X_1), 0, s(0)), p_2(s(X_1), s^2(0), 0)$$

Hence, to obtain cellular automata for a larger set of NLPs, one would have to use a more general model of cellular automata, such as the one proposed by (Blair et al 1997). However, that is out of scope of this paper.

5 Decidability Results

Now that we have derived a construction between logic programs and cellular automata, we are able to view problems from cellular automata theory as problems of logic programming and, by extension, first-order logic. We now show how some computational problems can be translated to first-order logic programs, and which conclusions we can draw from this. Before we can examine these problems, we first give definitions of well-known objects from cellular automata theory, as well as a definition of Turing machines that will be used in various proofs. Note that some results about the decidability of infinite behavior of general (i.e., non-OB) cellular automata have been found in (Packard and Wolfram 1985). Here, we provide new (but more specific) theorems and proofs on OBCAs as illustration of how our construction can be used to effortlessly solve problems in CA theory. At the same time, we show new results on properties of orbits with respect to the $\mathcal{T}_\mathcal{P}$ operator of NLPs, including the existence of supported models (Apt, Blair, and Walker 1988) and supported sets (Inoue and Sakama 2012).

Definition 3
A *still life* is a configuration c of a cellular automaton which is a fixpoint of the transition function δ, that is, a configuration fulfilling $\delta(c) = c$.

Note that a fixpoint of δ translates to a fixpoint of the $T_\mathcal{P}$ operator, i.e., a supported model of the logic program (Inoue 2011).

Definition 4
An *oscillator* is a sequence of configurations $(c_1, \ldots, c_n)$ of a cellular automaton that form an *attractor* of the transition function δ, that is, they fulfill $\delta(c_i) = c_{i+1}$ for $i < n$, and $\delta(c_n) = c_1$.

The corresponding attractor of the $T_\mathcal{P}$ forms a *supported class* of the logic program (Inoue and Sakama 2012).

Definition 5
A *spaceship* is an infinite sequence of configurations $(c_i)_{i \in \mathbb{N}^+}$ a *period* $n \in \mathbb{N}^+$ and a *translation* $t \in \mathbb{N}^d$ of a cellular automaton with the following properties:

- $\delta(c_i) = c_{i+1}$
- $c_i + t = c_{i+n}$

A classic example of a spaceship is the *glider* in Game of Life (Gardner 1970). As far as the authors know, the corresponding concept of spaceships has never been formalized in the semantics of logic programming. In the following sections, we will use the names oscillator and spaceship also for any configuration that is part of an oscillator or a spaceship.

5.1 Turing Machine

There are many largely equivalent definitions of a Turing machine. We will use the following definition:

Definition 6
A *one-way bounded Turing Machine* (OBTM) consists of a quintuple $(Q, \Sigma, q_0, \square, \delta)$, where Q is the set of states, Σ is the tape alphabet, fixed to $\Sigma = \{0, 1, \square\}$, q_0 is the initial state, $\square$ is the blank tape symbol, and $\delta : Q \times \Sigma \to Q \times \Sigma \times \{\leftarrow, \downarrow, \rightarrow\}$ is the transition function. The tape of this Turing machine is bounded to the left and infinite to the right, much like the cell space of a one-dimensional OBCA. Furthermore, we define that the machine halts if a transition is taken where the direction of the head movement is $\downarrow$.

Since logic programs and Turing machines are both computationally universal, it is possible to encode one as the other. However, here it is important to guarantee that encoding of a Turing machine into an NLP can be done within the class of NLPs that can be translated to cellular automata.

5.2 Orbit Problems

The orbit problems we are considering have the form: *Given an OBCA $\mathcal{A}$ and a configuration c, does the orbit of c contain a configuration that exhibits a property P?*

Theorem 2
The orbit problem of still lives is undecidable.

Proof
Given an OBTM $\mathcal{M}$, construct the corresponding logic program and from that the corresponding OBCA. Observe that the head always moves, unless the OBTM halts. The OBTM reaches a still life iff it halts. Therefore, the orbit of a configuration c leads to a still life iff $\mathcal{M}$ halts on c. Therefore, if there was an algorithm to decide the orbit problem of still lives, this algorithm could be used to decide the halting problem, which is known to be undecidable (Turing 1937). $\square$

Corollary 1
The question whether an orbit of the $T_\mathcal{P}$ operator leads to a supported model is undecidable.

Theorem 3
The orbit problem of spaceships is undecidable.

Proof
Given an OBTM $\mathcal{M}$, construct an OBTM $\mathcal{M}'$ that does the following:

- Move the input one cell to the right;
- Write a 1 to the now free first cell;
- Simulate $\mathcal{M}$ on the moved input;
- If $\mathcal{M}$ terminates, clear the tape and simulate any spaceship.

If $\mathcal{M}$ terminates, the orbit of the starting configuration on $\mathcal{M}'$ leads into a spaceship. If $\mathcal{M}$ does not terminate, the first cell of the tape never clears, which means that the whole configuration does not perform a translation, so there is no spaceship. Therefore, if there was an algorithm to decide the orbit problem of spaceships, this algorithm could be used to decide the halting problem. □

Similar arguments can be made for oscillators or other orbit problems, therefore:

Theorem 4
The question whether an orbit of the $T_{\mathcal{P}}$ operator leads to an attractor is undecidable.

The other two problems are the testing problem (*Given an OBCA $\mathcal{A}$ and a configuration c, does c exhibit a property P?*) and the existence problem (*Given an OBCA $\mathcal{A}$, is there a configuration c that exhibits a property P?*).

6 Perspectives

There is a variety of avenues in which research still is ongoing. First of all, one of the most restrictive conditions in the given construction lies in the translation from logic programs to cellular automata. Since only a limited class of logic programs is captured by that construction, we cannot readily use it to perform proofs or create models for general logic programs; therefore, we are interested in a more general class of cellular automata that is capable of modeling a wider range of logic programs. Since such a construction inevitably sacrifices the grid-based cellular automaton model as well as a notion of locality, it remains to be seen whether there is a general structure that retains some useful properties, or if the general automaton model is so unspecific that it has no value over using the logic program directly. Another possible result of research in this area is obtaining an automaton model that is not general, but more powerful than the given OBCAs and UCAs, for instance with a tree-based structure to transcend the carthesian cell space.

Going in the other direction, while OBCAs are ideal to model this restricted class of logic programs, the classical notion of cellular automata (for instance, Conway's Game of Life) utilizes an unbounded grid, as described in the UCA structure. Problems in this area include the fact that the application depth of a function to a constant is a natural number, while the coordinates in a UCA are integers. Although there exist several bijections between the natural and integer sets, it is

difficult to use one of these bijections properly, as we have to preserve the notion of locality, that is, whether an integer is positive or negative, its neighborhood (as modelled by different application depths in the NLP) has to be the same. Other approaches, such as using one function for positive and one for negative integers, have the problem that only a subset of all Herbrand interpretations translate to legal configurations of the CA. Therefore, existence proofs on the logic program do not trivially correspond to existence proofs on the cellular automaton.

References

Apt, K. R.; Blair, H. A.; and Walker, A. Towards a theory of declarative knowledge. In: *Foundations of Deductive Databases and Logic Programming*, Morgan Kaufmann, pp.89–148, 1988.

Blair, H. A., Dushin, F., and Humenn, P. R. Simulations between programs as cellular automata. In: Dix, J.; Furbach, U.; and Nerode, A., eds., *LPNMR*, volume 1265 of *Lecture Notes in Computer Science*, pp.115–131, Springer, 1997.

Blair, H. A., Chidella, J., Dushin, F., Ferry, A., and Humenn, P. R. A continuum of discrete systems. *Annals of Mathematics and Artificial Intelligence*, 21:153–186, 1997.

Gardner, M. Mathematical games—the fantastic combinations of John Conway's new solitaire game "Life". *Scientific American*, 223:120–123, 1970.

Grädel, E. Decidable fragments of first-order and fixed-point logic—from prefix vocabulary classes to guarded logics. 2003.

Inoue, K. Logic programming for Boolean networks. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pp.924–930, 2011.

Inoue, K. and Sakama, C. Oscillating behavior of logic programs. In: Erdem, E.; Lee, J.; Lierler, Y.; and Pearce, D., eds., *Correct Reasoning—Essays on Logic-Based AI in Honour of Vladimir Lifschitz*, volume 7265 of *Lecture Notes in Computer Science*, pp.345–362, Springer, 2012.

Kari, J. Theory of cellular automata: A survey. *Theoretical Computer Science*, 334(1-3):3–33, 2005.

Kauffman, S. A. *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, 1993.

Langton, C. G. Studying artificial life with cellular automata. *Physica D*, 2(1-3):120–149, 1986.

Packard, N. and Wolfram, S. Two-dimensional cellular automata. *Journal of Statistical Physics*, 38(5-6):901–946, 1985.

Sakama, C. and Inoue, K. Abduction, unpredictability and Garden of Eden. *Logic Journal of the IGPL* 21(6):980–998, 2013.

Subrahmanian, V. S. On the semantics of quantitative logic programs. In: *Proc. SLP*, pp.173–182, IEEE Computer Society, 1987.

Syrjänen, T. Modelling the game of life using logic programs. In: Leksa Notes in Computer Science: *Festschrift in Honour of Professor Leo Ojala*, Helsinki University of Technology, 2000.

Turing, A. On computable numbers, with an application to the Entscheidungsproblem. In: *Proceedings of the London Mathematical Society*, pp.230–265, 1937.

Wolfram, S. Computation theory of cellular automata. *Communications in Mathematical Physics*, 96(1):15–57, 1984.

Wolfram, S. *A New Kind of Science*. Wolfram Media Inc., 2002.