Complex Systems **1** (1987) 69–80

Formal Language Characterizations of Cellular Automaton Limit Sets

Lyman P. Hurd
Mathematics Department, Princeton University,
Princeton, NJ 08544, USA

Abstract. A formal language description of one-dimensional cellular automata limit sets is given, and a series of examples illustrating several degrees of complexity are constructed. The undecidability of membership of a string in the limit set of a cellular automaton rule is proven.

1. Introduction

Cellular automata are simple dynamical systems in which a discrete lattice of sites taking on values in a finite set evolves in accordance with a local rule. This paper provides an answer to some of the problems raised by Wolfram in [1], and provides a framework for studying cellular automata in terms of formal language theory. The first section defines the language associated with a cellular automaton limit set. The second provides an overview of the definitions and results needed from language theory, and the third gives a series of cellular automata whose limit sets have increasing complexity. The fourth provides the proof to a technical lemma needed in the previous section and the fifth gives general conclusions.

For the purposes of this paper a cellular automaton will consist of a one dimensional lattice of sites and a transition rule depending only on the value of a site and a finite number of neighbors. Sites take values in a finite alphabet S with k elements. The value assumed by a given site under the transition rule will depend on sites of up to r steps away.

Formally, a *cellular automaton* consists of a finite set of site values S and a local transformation

$$\phi : S^{2r+1} \to S$$

from which one constructs a global transition rule $\Phi : S^Z \to S^Z$ given by $\Phi(s)_n = \phi(s_{n-r}, ..., s_0, ..., s_{n+r})$. This construction yields all continuous functions from S^Z to itself (with S given the discrete topology, and S^Z the product topology) which are translation-invariant (commute with the shift map) [2].

2. Cellular automata limit sets

The *limit set* associated with a cellular automaton consists of those configurations which have past histories of arbitrary length.

Given a rule Φ let $\Omega_\Phi^{(0)} = S^Z$, and $\Omega_\Phi^{(i)} = \Phi(\Omega_\Phi^{(i-1)})$ the image of S^Z under i iterations of Φ. Notice that $\Omega_\Phi^{(i)} \supseteq \Omega_\Phi^{(j)}$ for all $i \leq j$. The set $\Omega_\Phi = \bigcap_{i=0}^{\infty} \Omega_\Phi^{(i)}$ is called the *limit set* of Φ.

Note that an equivalent condition for a configuration c to be in the limit set is that there exist a countably infinite collection of configurations c^{-i} such that $c^0 = c$ and $\Phi(c^{-i}) = c^{-i+1}$.

3. Limit languages of cellular automata

Since any device attempting to make decisions about the limit set of a cellular automata in finite time can only scan a finite number of symbols, it makes sense to deal with sets of configurations in terms of their finite substrings. This reasoning motivates the following:

Definition 1. *The set of all finite connected substrings of a set of configurations $\Omega \subseteq S^Z$ is called the language associated with Ω, denoted $\mathbf{L}[\Omega]$. In particular, if Φ is a cellular automaton rule, $\mathbf{L}[\Omega_\Phi]$ is called the limit language of Φ.*

The fact that the set of finite strings determines the set of configurations follows from:

Theorem 1. *If Δ_1 and Δ_2 are closed, translation-invariant subsets of S^Z, then $\mathbf{L}[\Delta_1] = \mathbf{L}[\Delta_2] \Rightarrow \Delta_1 = \Delta_2$.*

Proof.

We show that $\Delta_1 \subseteq \Delta_2$.

Given $c \in \Delta_1$, define a family of finite substrings of c by letting

$$c_{(i)} = c_{-i} \ldots c_0 \ldots c_i$$

Each $c_{(i)} \in \mathbf{L}[\Delta_1]$ therefore $c_{(i)} \in \mathbf{L}[\Delta_2]$ by assumption.

Let $A_i = \{d \in S^Z | d_{(i)} = c_{(i)}\}$ and $A_\infty = \bigcap_{i=0}^{\infty} A_i$ By definition, A_∞ consists of the single element c. Let $B_i = A_i \bigcap \Delta_2$ and $B_\infty = \bigcap_{i=0}^{\infty} B_i$.

The A_i are closed; therefore the B_i are closed. Since $c_{(i)} \in \mathbf{L}[\Delta_2]$, $B_i \neq \emptyset$ for any i; therefore, by compactness, $B_\infty \neq \emptyset$. Therefore since $B_\infty \subseteq A_\infty$ and $B_\infty \neq \emptyset$, $B_\infty = A_\infty \subseteq \Delta_2$ and therefore $c \in \Delta_2$. Thus $\Delta_1 \subseteq \Delta_2$. Similarly $\Delta_2 \subseteq \Delta_1$. Therefore $\Delta_1 = \Delta_2$. ∎

Corollary 1. *Given two rules Φ_1 and Φ_2, $\mathbf{L}[\Omega_{\Phi_1}] = \mathbf{L}[\Omega_{\Phi_2}] \Rightarrow \Omega_{\Phi_1} = \Omega_{\Phi_2}$.*

$T = \{a, b\}$
$N = \{\Sigma, A\}$
$\Sigma \rightarrow A$
$A \rightarrow \epsilon$(empty string)
$A \rightarrow aAb$

Table 1: A generative grammar for the language $a^n b^n$.

4. Formal languages [1]

Formal language theory provides a framework within which to study sets of strings from a finite alphabet. Languages may be viewed either as outputs of some class of machines, or as the end product of typographical substitution systems (generative grammars).

The basic machine model for these languages and indeed for much of computation is the *Turing machine*. The machines which recognize each of the families of languages are described as restrictions of full Turing machines.

A Turing machine T consists of a tape divided into squares each of which contains a symbol from an alphabet A, and a head which occupies some position along the tape and which is in one of a finite set of states Q. In each time step the head writes a symbol in the square it occupies and possibly moves either right or left all depending on its current state and the symbol in the square it currently occupies.

Formally, a *Turing Machine* is determined by a set of states Q, an alphabet A and three functions, the output function $F : Q \times A \rightarrow A$, the state transition function, $G : Q \times A \rightarrow Q$ and the head direction function $D : Q \times A \rightarrow \{-1, 1\}$ which determines whether the head will move to the left or to the right.

Another way of describing formal languages is by means of a *generative grammar*. Such a grammar consists of a start symbol, Σ, a set of terminal symbols, T, a set of non-terminal symbols, N, and a set of generative rules, represented by typographic substitutions. The language corresponding to such a grammar consists of all strings of terminals which may be derived from the start symbol by successive application of the production rules. An example of such a grammar is shown in table 1. This illustration shows the grammar corresponding to the language $a^n b^n$.

Recursively enumerable languages can be divided into the *Chomsky Hierarchy* based on the complexity of the grammar which produces them, or equivalently, the complexity of the machine needed to recognize them. This hierarchy is shown in Table 2.

At the top of the hierarchy are the *recursively enumerable (r.e.)* languages. This class consists of all languages whose strings may be produced as the output of a general Turing Machine. Languages in this class are generated by unrestricted generative grammars. An example of such a lan-

[1]The background material in this section can be found in [4] and [5].

Language	Grammar	Machine
r.e. languages	unrestricted	Turing Machine
context-sensitive	string $\rightarrow$ longer string or $\Sigma \rightarrow \epsilon$ (empty string)	linear bounded automaton
context-free	symbol $\rightarrow$ string	push down automaton
regular	right linear non-terminal $\rightarrow$ non-terminal. terminal or non-terminal $\rightarrow$ terminal	finite state machine

Table 2: The Chomsky hierarchy of formal languages.

guage is given by strings of the form x^h where h is the number of a halting Turing Machine in some fixed enumeration.

A smaller class of languages are the *context-sensitive* languages, which have generative grammars with the property that every production rule increases the length of the string. The words in such a language are recognized by *linear bounded automata* which are Turing machines whose memory is bounded by a linear function of the length of the word to be recognized. An example of a context-sensitive language is the set of strings over the alphabet $\{x, y, z\}$ of the form $x^n y^n z^n$.

Another example of a context sensitive language, is the language consisting of valid runs for a Turing machine. A program history can be checked in bounded space, because, in n time steps, the head cannot have visited more than n distinct squares.

Simpler than context-sensitive languages, are the *context-free* languages whose grammars have the property that the only strings which may appear on the left side of transformation rules have length one. They are recognized by *push down automata*, which are Turing machines which have a stack (first in, last out memory) instead of a tape. The language of balanced parentheses is context-free.

All of the previous types of language potentially require an infinite memory capacity. *Regular languages* consist of those languages which can be recognized by machines with a finite amount of memory, *finite state machines*. They are generated by *right linear* (or *left linear*) grammars, which have the form that a non-terminal is sent to a non-terminal followed by a terminal or to a terminal.

Equivalently, words in a regular language can be represented as walks through a graph whose edges are labeled with symbols from the given alphabet. An example of a regular language is the set of all strings in the alphabet $\{0, 1\}$ which do not contain two consecutive ones. The graph to which this corresponds is shown in figure 1. One can assume without loss of generality that the graph has no more than one edge of a given label leaving each node (the finite state machine in this case is said to be *deterministic*). By the Myhill-Nerode Theorem, there exists a canonical, minimal such graph. The number of nodes in the minimal graph provides

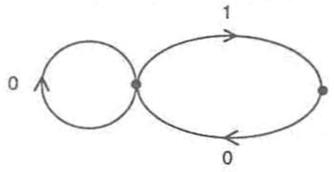

Figure 1: Graph representing a regular language containing all strings in the alphabet $\{0, 1\}$ with no two consecutive ones.

a measure of complexity for the regular language.

A third way of representing strings in a regular language is by means of a *regular expression*. A regular expression is either the empty string ϵ, a symbol from the alphabet, the composition of two regular expressions under the operations of disjunction "+" and concatenation ".", or the *Kleene closure* of a regular expression (any finite number of occurrences of the expression) *expression** . For example, all strings in the alphabet $\{0, 1\}$ which do not contain two consecutive ones satisfy the expression $(0+1)(00+01)^*$. Set braces around a regular expression will be used to denote the set of strings satisfying the given expression.

The position of a language in this hierarchy has a bearing on the kinds of propositions that can be answered about it. For example, for many classes of languages, the question of whether two grammars determine the same language, is in general undecidable.

5. Determining language complexity

Given a generative grammar, it is in general undecidable which step of the hierarchy it occupies, unless it is given in one of the restricted forms listed. There are, however, a sequence of lemmas which in some cases guarantee that a given language *is not* in a given class. These are the *pumping lemmas*.

The pumping lemma for regular languages states that for every regular language R, there exists a number n such that every string of length greater than or equal to n can be written as the concatenation of three strings, abc such that the string ab^ic is also in R for all i. If one looks one strings in a regular language as labelled walks through a finite graph, this is a restatement of the observation that every sufficiently long path through a finite graph must contain a cycle. Thus if one can show that a given language has a set of strings which do not satisfy the lemma, it cannot be regular. For example, the language a^nb^n is not regular.

Similarly there is a pumping lemma for context-free languages. It states that given a context-free language F, there is an n such that every string of length greater than or equal to n can be written as the concatenation of five strings $abcde$ such that ab^icd^ie is in F for all i. Thus, for example, the

$S = \{r, l, W, o\}$ and $r = 2$.		
$.rox.$	$\rightarrow$	r
$..rox$	$\rightarrow$	o
$..yol$	$\rightarrow$	l
$yol..$	$\rightarrow$	o
$..rWl$	$\rightarrow$	l
$rWl..$	$\rightarrow$	r
otherwise the identity rule		

Table 3: The cellular automaton rule Λ, which has a non-regular limit language. x denotes and symbol except l. y denotes any symbol except r. . denotes any symbol.

language $a^n b^n c^n$ is not context-free.

6. Cellular automaton limit languages

This section contains a series of examples of rules whose limit languages are strictly more complicated than a given language class. There are many examples of rules whose limit languages can be proven to be regular. The question remains open whether there are rules whose limit languages are strictly context-free or strictly context-sensitive. Since the complexity of a language generated by an arbitrary generative grammar is undecidable, this question is likely to be somewhat difficult.

Wolfram has shown [1] that $\mathbf{L}[\Omega_\Phi^{(n)}]$ is a regular language for any cellular automaton rule Φ. A question which naturally presents itself is whether $\mathbf{L}[\Omega_\Phi]$ is regular for every rule Φ. This paper shows that this is not the case. In fact, the limit language of a rule need not even be recursively enumerable.

6.1 A cellular automaton with a non-regular limit language [2]

In this example, a rule Λ is given whose limit language is more complicated than a regular language.

This rule has $S = \{r, l, W, o\}$ and $r = 2$. The transition rules are given in table 3.

The evolution of this rule from a sample initial state is shown in figure 2.

Theorem 2. *The language* $\mathbf{L}[\Omega_\Lambda]$ *is not regular.*

Proof. The intersection of any two regular languages is always a regular language. Therefore, it suffices to produce a regular language whose intersection with $\mathbf{L}[\Omega_\Lambda]$ is not regular.

Consider $F = \mathbf{L}[\Omega_\Lambda] \bigcap \{oolo^*Wo^*roo\}$

[2]The basic idea behind this rule was suggested by [6].

```
W   l   r   W     l     W     l          l
W  l     r  W    l      W    l          l
W l       r W   l       W   l          l
Wl         rW  l        W  l          l
Wl         rW l         W l          l
Wl         rWl          Wl          l
Wl         lWr          Wl         l
Wl        l W r         Wl        l
Wl       l  W  r        Wl       l
Wl      l   W   r       Wl      l
Wl     l    W    r      Wl     l
Wl    l     W     r     Wl    l
Wl   l      W      r    Wl   l
Wl  l       W       r   Wl  l
Wl l        W        r  Wl l
Wll         W         r Wll
Wll         W          rWll
Wll         W          lWrl
Wll         W         l Wrl
Wll         W        l  Wrl
Wll         W       l   Wrl
Wll         W      l    Wrl
Wll         W     l     Wrl
Wll         W    l      Wrl
Wll         W   l       Wrl
Wll         W  l        Wrl
Wll         W l         Wrl
Wll         Wl          Wrl
Wll         Wl          Wrl
Wll         Wl          Wrl
Wll         Wl          Wrl
Wll         Wl          Wrl
```

Figure 2: The evolution of cellular automaton rule Λ from a sample initial state. Blanks denote the symbol o.

$S = \{r, R, l, L, W, o\}$ and $r = 4$		
...roxy..	$\rightarrow$	r
....roxy.	$\rightarrow$	o
..uvol...	$\rightarrow$	l
.uvol....	$\rightarrow$	o
..Rooxy..	$\rightarrow$	R
....Rooxy	$\rightarrow$	o
..uvooL..	$\rightarrow$	L
uvooL....	$\rightarrow$	o
...RrWlL.	$\rightarrow$	l
....RrWlL	$\rightarrow$	L
.RrWlL...	$\rightarrow$	r
RrWlL....	$\rightarrow$	R
...rRWLl.	$\rightarrow$	L
....rRWLl	$\rightarrow$	l
.rRWLl...	$\rightarrow$	R
rRWLl....	$\rightarrow$	r
otherwise the identity rule		

Table 4: The cellular automaton rule Λ', which has a non-context-free limit language. x denotes any symbol except l or L. y is any symbol except L. v is any symbol except r or R. u is any symbol except R. . stands for any symbol.

First $F \neq \emptyset$. The string $oolo^nWo^nroo$ has a family of predecessors given by $o^*ro^iWo^ilo^*$. It remains to show that these strings exhaust F.

Let $c = oolo^nWo^mroo$ be a string in the intersection. If $n, m > 0, c$ has a predecessor of the form $oolo^{n-1}Wo^{m-1}roo$. Since $oolWo^sroo$ has no predecessor when $s > 0$, $n = m$.

The language $\{oolo^nWo^nroo\}$ violates the pumping lemma for regular languages. Thus F is not a regular language, and therefore $\mathbf{L}[\Omega_\Lambda]$ is not. ∎

6.2 A cellular automaton rule with a non-context-free limit language

By generalizing the previous construction one can construct a cellular automaton rule Λ' whose limit language is not context-free. The strategy is essentially the same. The intersection of a regular language and a context-free language yields a context-free language. One finds a regular language whose intersection with $\mathbf{L}[\Omega_{\Lambda'}]$ is not context-free. The evolution of this rule from a sample initial state is shown in figure 3.

Theorem 3. *The language $\mathbf{L}[\Omega_{\Lambda'}]$ is not context-free.*

Proof. Let $F' = \mathbf{L}[\Omega_{\Lambda'}] \bigcap \{ooooLo^*lo^*Wo^*ro^*Roooo\}$.

Once again $F' \neq \emptyset$. Every string of the form $ooooLo^{n+2}lo^nWo^nro^{n+2}Roooo$ has a family of predecessors of the form $o^*Ro^sro^sWo^slo^sLo^*$.

```
 RrW   1     L WR     r     W 1  L         1
 RrW  1    L   W  R    r    W1 L          1
 RrW 1   L     W    R   r   W1L          1
 RrW1  L       W      R  r  W1L         1
 RrW1L         W        R r W1L        1
 L1WrR         W        R  rW1L       1
  1Wr  R       W          RrW1L      1
 1 W r   R     W          L1WrR     1
1  W  r    R   W        L  1Wr R   1
   W   r      RW      L   1 W r  R1
   W    r     RW    L    1  W  r R1
   W     r    RW  L     1   W  r R1
   W      r   RWL      1    W   rR1
   W       r  RWL     1     W   rR1
   W        r RWL    1      W   rR1
   W         rRWL   1       W   rR1
   W         rRWL  1        W   rR1
   W         rRWL 1         W   rR1
   W         rRWL1          W   rR1
   W         L1WrR          W   rR1
   W       L  1Wr  R        W   rR1
   W     L   1 W r   R      W   rR1
   W   L    1  W  r    R    W   rR1
   W L     1   W   r     R  W   rR1
   WL     1    W    r      RW   rR1
   WL    1     W     r     RW   rR1
   WL   1      W      r    RW   rR1
   WL  1       W       r   RW   rR1
   WL 1        W        r  RW   rR1
   WL1         W         r RW   rR1
   WL1         W          rRW   rR1
   WL1         W          rRW   rR1
```

Figure 3: The evolution of cellular automaton rule Λ' from a sample initial state. Blanks denote the symbol o.

$S = \{r, l, q_1, ..., q_n\} \times \{s_1, ..., s_m\}$			
$D(q_n, s_i) = 1$	$(q_n, s_i)(r, s_j)*$	$\rightarrow$	$(G(q_n, s_i), s_j)$
	$*(q_n, s_i)*$	$\rightarrow$	$(l, F(q_n, s_i))$
$D(q_n, s_i) = -1$	$*(l, s_j)(q_n, s_i)$	$\rightarrow$	$(G(q_n, s_i), s_j)$
	$*(q_n, s_i)*$	$\rightarrow$	$(r, F(q_n, s_i))$
otherwise the identity rule			

Table 5: The cellular automaton rule Ψ_T, which corresponds to any given Turing Machine. $*$ denotes any symbol.

Every expression of the form $ooooLo^ilo^jWo^kro^lRoooo$ has a predecessor containing the string $ooooLo^{i-1}lo^{j-1}Wo^{k-1}ro^{l-1}Roooo$ with $i, j, k, l > 0$. Now $ooooLl$ has no predecessor, so $i \geq j$ and $l \geq k$. Furthermore, the only context in which lWr may appear after the first time step, is $LlWrR$ which is the case above. Hence these strings exhaust the intersection.

F' violates the pumping lemma for context-free languages. Since F' is not a context-free language, $\mathbf{L}[\Omega_{\mathbf{A}'}]$ is not. ∎

6.3 A cellular automaton rule with a non-recursive limit language

A Turing machine T as defined earlier was specified by a quintuple $\{Q, A, F, G, D\}$ in which $Q = \{q_1, ..., q_n\}$ is a set of states, $A = \{s_1, ..., s_m\}$ is an alphabet of symbols, and $F : Q \times A \rightarrow A$, $G : Q \times A \rightarrow Q$, and $D : Q \times A \rightarrow \{-1, 1\}$ are the output, state transition, and head direction functions respectively. Given a Turing Machine T, we define a cellular automata rule Ψ_T which emulates it. The square which the head occupies will be represented by an ordered pair (q_i, s_j) where $q_i \in Q$ indicates the state of the head, and $s_i \in A$ indicates the tape symbol. Squares not occupied by the head are of the form (r, s_i) or (l, s_i) indicating that the square is to the right or left of the head, and has symbol s_i . The constraint that the head only writes squares which are on the correct side assures that no square can be rewritten by two different heads. A head attempting to move into another head or a square which is not on the appropriate side, disappears.

Theorem 4. *Given a Turing machine T and a state q_h , it is in general undecidable whether (q_h, s_i) is in the limit language of Ψ_T.*

Proof. The question whether (q_h, a_i) is in the limit language is equivalent to the question whether there exist legal runs of arbitrary length which leave T in state q_h. This question is undecidable by the following lemma.

Lemma 1. *Given a Turing machine T and a state q_h is in general undecidable whether T can assume state q_h after n time steps for any n.*

A proof of this result is sketched in [7] and proven in full in [8]. A proof is sketched in section 5.

Theorem 5. *The complement of* $\mathbf{L}[\Omega_\Psi]$ *is r.e. for all rules* Ψ.

Proof. The complement of the limit language of a cellular automata is always recursively enumerable. If a string s is in the complement of the limit language, it must be excluded after some number of time-steps n. The strings of length m excluded by the n^{th} time-step may be computed by calculating the image under the n^{th} iterate of Φ of all strings of length $m + 2nr$.

Corollary 2. *There exists a rule* Φ *such that the language* $\mathbf{L}[\Omega_\Phi]$ *is not recursively enumerable.*

A language is recursive if and only if both it and its complement are r.e. We have shown that there exists a rule whose limit language is not recursive, thus it is not r.e.

7. Turing Machine state histories

Proof of lemma. The problem of whether a Turing machine can attain a given state after an arbitrary number of time steps can be reduced to the problem of a machine halting on a blank tape, a known undecidable problem.

Definition 2. *Given a Turing machine* $T = \{Q, A, F, G, D\}$ *a state history for* T *consists of a finite sequence of states* $q_0, q_1, ..., q_n$ *such that if* T *is started on a blank tape in state* q_0*, it will proceed in order to states* $q_1, q_2, ... q_n$.

The *halting problem on a blank tape* which is undecidable, is equivalent to the statement that the machine T has state histories of arbitrary length.

The reduction proceeds as follows. Given an arbitrary Turing Machine T we construct a new machine T' which recognizes state histories for T. In particular if T' is started on a tape containing the symbols $Lq_0, q_1, ..., q_n R$, it checks that $q_0, q_1, ..., q_n$ represents a valid state history for T, and if it does, it writes an R over q_n and repeats the procedure. Otherwise it enters an alarm state. The machine can be constrained in such a way that the only case in which a non-alarm state can occur after an arbitrary number of time steps, is that there are state histories of arbitrary length for T , i.e. T does not halt. But this question is undecidable, hence the question whether T' can be in a given state after arbitrary time is undecidable.

8. Discussion

At present only a very few limit languages have been computed explicitly. This is not unexpected. According to Rice's Theorem, any non-trivial (not always true or always false) proposition about r.e. languages is in general undecidable. An analogous result is conjectured to hold true in the case of cellular automata. This would mean that in particular that it is undecidable whether the limit language corresponding to a given rule is regular, consists of more than one string, or in general if two cellular automata yield the same limit set.

Work needs to be done to characterize exactly which languages can occur as limit languages for cellular automata. Even the weaker question of which regular languages can occur as the 1 time step image of a cellular automaton is unsolved.

Acknowledgements

I am grateful for support from Thinking Machines Corporation in the early stages of this work, and for discussions with T. Hales, D. Hillis, B. Kuszmaul, J. Milnor, N. Packard, and S. Wolfram.

References

[1] S. Wolfram, "Computation theory of cellular automata", *Communications in Mathematical Physics,* **96** (1984) 15-57.

[2] G. A. Hedlund, "Endomorphisms and automorphisms of the shift dynamical system", *Mathematical Systems Theory* **3** (1969) 320; G. A. Hedlund, "Transformations commuting with the shift", in *Topological dynamics*, ed. J. Auslander and W. H. Gottschalk, (Benjamin, 1968).

[3] P. Walters, *An introduction to ergodic theory*, (Springer, 1982).

[4] J. E. Hopcroft and J. D. Ullman, *Introduction to automata theory, languages, and computation*, (Addison-Wesley, 1979).

[5] M. Minsky, *Computation: finite and infinite machines*, (Prentice-Hall, 1967).

[6] D. Hillis private communication.

[7] U. Golze, "Differences between 1- and 2-dimensional cell spaces", in A. Lindenmayer and G. Rozenberg (eds.), *Automata, languages, development*, (North-Holland, 1976).

[8] U. Golze, "Endliche, Periodische, und Rekursive Zellulare Konfigurationen : Vorgangerberechnung ung Garten-Eden-Probleme", dissertation to der Technischen Universitat Hannover.