Alexander Okhotin

An 11-state trellis automaton for a P-complete language

Turku Centre *for* Computer Science

TUCS Technical Report
No 850, October 2008

An 11-state trellis automaton for a P-complete language

Alexander Okhotin
Department of Mathematics, University of Turku, *and*
Turku Centre for Computer Science
Turku FIN–20015, Finland, *and*
Academy of Finland
alexander.okhotin@utu.fi

TUCS Technical Report
No 850, October 2008

Abstract

An 11-state trellis automaton recognizing a P-complete language over the alphabet $\{a, b\}$ is constructed.

Keywords: trellis automata, cellular automata, conjunctive grammars, computational complexity, circuit value problem

TUCS Laboratory
Discrete Mathematics for Information Technology

1 Introduction

Trellis automata are one of the simplest, perhaps *the* simplest kind of cellular automata, and are known as *one-way real-time cellular automata* in the standard nomenclature. The first results on their expressive power are due to Smith [11], Dyer [2] and Culik et al. [1]. By definition, a trellis automaton uses space n and makes $\Theta(n^2)$ transitions, so every language it recognizes is in P. The existence of a trellis automaton accepting a P-complete language was demonstrated by Ibarra and Kim [5], though no explicit construction was presented. A particular automaton solving a P-complete problem was constructed by the author [8]; it used 45 states and was defined over a 9-letter alphabet was given.

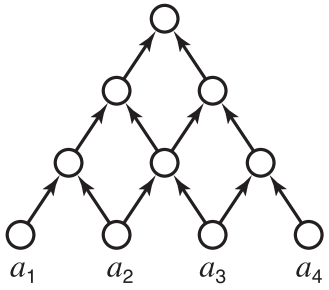
This paper aims to construct a new trellis automaton solving a different P-complete problem, this time with the goal of minimizing the number of states. The problem is the same variant of the Circuit Value Problem as in the previous paper [10], though this time a new encoding is defined. With the proposed encoding, the problem may be solved by an 11-state trellis automaton over a 2-letter alphabet. A full construction will be given and explained.

2 Trellis automata

Trellis automata can be equally defined by their cellular automata semantics (using evolution of configurations) and through the trellis representing their computation. According to the latter approach, due to Culik et al. [1], a trellis automaton processes an input string of length $n \geqslant 1$ using a uniform triangular array of $\frac{n(n+1)}{2}$ processor nodes, as presented in the figure below. Each node computes a value from a fixed finite set Q. The nodes in the bottom row obtain their values directly from the input symbols using a function $I : \Sigma \to Q$. The rest of the nodes compute the function $\delta : Q \times Q \to Q$ of the values in their predecessors. The string is accepted if and only if the value computed by the topmost node belongs to the set of accepting states $F \subseteq Q$. This is formalized in the following definition.

Definition 1. *A trellis automaton is a quintuple $M = (\Sigma, Q, I, \delta, F)$, where:*

- *Σ is the input alphabet,*
- *Q is a finite non-empty set of states,*
- *$I : \Sigma \to Q$ is a function that sets the initial states,*
- *$\delta : Q \times Q \to Q$ is the transition function, and*
- *$F \subseteq Q$ is the set of final states.*



The result of the computation on a string $w \in \Sigma^+$ is denoted by $\Delta : \Sigma^+ \to Q$, which is defined inductively as $\Delta(a) = I(a)$ and $\Delta(awb) = \delta(\Delta(aw), \Delta(wb))$, for any $a, b \in \Sigma$ and $w \in \Sigma^*$. Then the language recognized by the automaton is $L(A) = \{w \mid \Delta(w) \in F\}$.

3 Sequential NOR Circuit Value Problem

A circuit is an acyclic directed graph, in which the incoming arcs in every vertex are considered ordered, every source vertex is labelled with a variable from a certain set $\{x_1, \ldots, x_m\}$ with $m \geqslant 1$, each of the rest of the vertices is labelled with a Boolean function of k variables (where k is its in-degree), and there is a unique sink vertex. For every Boolean vector of input values $(\sigma_1, \ldots, \sigma_m)$ assigned to the variables, the value computed at each gate is defined as the value of the function assigned to this gate on the values computed in the predecessor gates. The value computed at the sink vertex is the output value of the circuit on the given input.

The *Circuit Value Problem* (CVP) is stated as follows: given a circuit with gates of two types, $f_1(x) = \neg x$ and $f_2(x, y) = x \wedge y$, and given a vector $(\sigma_1, \ldots, \sigma_m)$ of input values assigned to the variables ($\sigma_i \in \{0, 1\}$), determine whether the circuit evaluates to 1 on this vector. The pair *(circuit, vector of input values)* is called an instance of CVP. This is the fundamental problem complete for P, which was proved by Ladner [6]. A variant of this problem is the *Monotone Circuit Value Problem* (MCVP), in which only conjunction and disjunction gates are allowed. As shown by Goldschlager [3], MCVP remains P-complete.

A multitude of other particular cases of CVP are known to be P-complete [4]. Let us consider one particular variant of this standard computational problem. A *sequential NOR circuit* is a circuit satisfying the following conditions:

- The notion of an input variable is eliminated, and the circuit is deemed to have a single source vertex, which, by definition, assumes value 1.
- A single type of gate is used. This gate implements *Peirce's arrow* $x \downarrow y = \neg(x \vee y)$, also known as the NOR function. It is well-known that every Boolean function can be expressed as a formula over this function only.
- The first argument of every k-th NOR gate has to be its direct predecessor, the $(k-1)$-th gate, while the second argument can be any previous gate. Because of that, these gates will be called *restricted NOR gates*.

The problem of testing whether such a circuit evaluates to 1 is called the *Sequential NOR Circuit Value Problem*, and it has recently been proved by the author [10] that it remains P-complete.

Theorem 1 ([10]). *Sequential NOR CVP is P-complete.*

The idea of the proof is to simulate unrestricted conjunction and negation gates by sequences of restricted NOR gates. An unrestricted negation gate of the form $C_i = \neg C_j$ can be simulated by two gates: $C_i = C_{i-1} \downarrow C_1$ and $C_{i+1} = C_i \downarrow C_j$. The gate C_1 is assumed to have value 1, so C_i will always evaluate to 0. Then C_{i+1} computes $\neg(0 \vee C_j) = \neg C_j$.

Similarly, a conjunction of C_j and C_k is represented by five restricted NOR gates: $C_i = C_{i-1} \downarrow C_1$, $C_{i+1} = C_i \downarrow C_j$, $C_{i+2} = C_{i+1} \downarrow C_1$, $C_{i+3} = C_{i+2} \downarrow C_k$ and $C_{i+4} = C_{i+3} \downarrow C_{i+1}$. Here C_i and C_{i+1} both evaluate to 0, C_{i+1} and C_{i+3} compute $\neg C_j$ and $\neg C_k$, respectively, and then the value of C_{i+4} is $C_j \wedge C_k$.

4 Encoding of circuits

Let us now give a simple encoding of sequential NOR circuits as strings over the alphabet $\Sigma = \{a, b\}$. Consider any sequential NOR circuit

$$\begin{aligned} C_1 &= 1 \\ C_2 &= C_1 \downarrow C_1 \\ C_3 &= C_2 \downarrow C_{j_3} \\ &\vdots \\ C_{n-1} &= C_{n-2} \downarrow C_{j_{n-1}} \\ C_n &= C_{n-1} \downarrow C_{j_n} \end{aligned}$$

where $n \geqslant 2$ and $1 \leqslant j_i < i$ for all i. The gates C_1 and C_2 are represented by strings a and b, respectively. Every restricted NOR gate $C_i = C_{i-1} \downarrow C_{j_i}$ with $i \geqslant 3$ is represented as a string ba^{j_i}. The whole circuit is encoded as a concatenation of these representations in the reverse order, starting from the gate C_n and ending with $\ldots C_3 C_2 C_1$. The encoding continues with a letter b and a suffix b^n representing the work space needed by the trellis automaton to store the computed values of the gates:

$$\underbrace{ba^{j_n}a^{j_{n-1}}\ldots ba^{j_4}ba^{j_3}ba}_{\text{gate descriptions}}\,b \quad \underbrace{b\ldots b}_{b^n\text{: work space}}$$

The set of syntactically correct circuit descriptions can be formally defined as follows:

$$L = \{ba^{j_n}\, ba^{j_{n-1}} \ldots ba^{j_3}\, b\, a\, b\, b^n \mid n \geqslant 2 \text{ and } 1 \leqslant j_i < i \text{ for each } i\}.$$

The language of correct descriptions of circuits that evaluate to 1 has the following fairly succinct definition:

$$\begin{aligned} L_1 = \{&ba^{j_n}ba^{j_{n-1}} \ldots ba^{j_3}bab\, b^n \mid n \geqslant 2 \text{ and } \exists x_1, x_2, \ldots, x_n, \text{ s.t.} \\ &x_1 = x_n = 1 \text{ and for all } i\ (1 \leqslant i \leqslant n),\ 1 \leqslant j_i < i \text{ and } x_i = \neg(x_{i-1} \vee x_{j_i})\}. \end{aligned}$$

This is a P-complete language and it has a simple structure that resembles the examples common in formal language theory. As it will now be demonstrated, this set can indeed be very succinctly defined by language-theoretic methods.

5 Construction of a trellis automaton

The goal is to construct a trellis automaton that accepts a string from L if and only if it is in L_1. Thus the behaviour of the automaton on strings from $\{a, b\}^+ \setminus L$ is undefined, and the actual language it recognizes is different from L_1. Disregarding the strings not in L results in a simpler construction and in fewer states.

The automaton uses 11 states, and its set of states is defined as $Q = \{?, 0^0, 0^1, 0^{\nwarrow}, 0^{\nearrow}, 0, 1^0, 1^1, 1^{\nwarrow}, 1^{\nearrow}, 1\}$. The initial function is defined by $I(a) = 0^{\nearrow}$ and $I(b) = 0^{\nwarrow}$, while the set of accepting states is $F = \{1\}$.

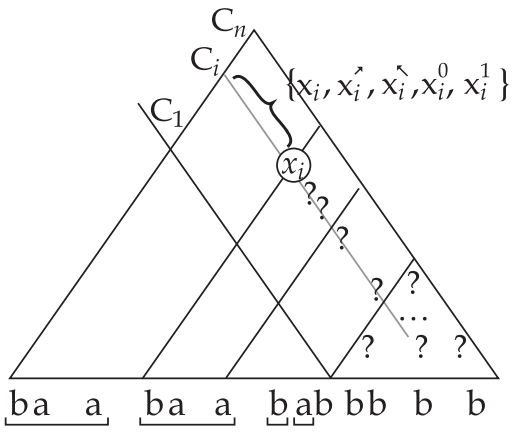


Figure 1: Sketch of the computation.

The overall structure of the computation of the automaton on a valid encoding of a circuit is given in Figure 1. The suffix b^n of the encoding is used by the automaton as the "work space", and the diagonal spawned to the left from every ith b in this suffix represents the computed value of the ith gate of the circuit. Each diagonal initially holds the question mark; in other words, $\Delta(wb^i) = ?$ for every sufficiently short nonempty suffix of the circuit description. The value of the ith gate is computed on the substring starting at the description of the ith gate and ending with b^i; formally,

$$\Delta(ba^{j_i}ba^{j_{i-1}} \dots ba^{j_3}babb^i) = \begin{cases} 0, & \text{if } C_i = 0; \\ 1, & \text{if } C_i = 1. \end{cases}$$

This computed value is propagated to the left, so that all subsequent states in this diagonal are $x^p \in Q$, where $x \in \{0, 1\}$ is the value of the gate C_i, while $p \in \{^0, ^1, ^{\nwarrow}, ^{\nearrow}, _{\sqcup}\}$ is a state of an ongoing computation of the trellis automaton.

In order to compute the value of each ith gate, the automaton should read the gate description ba^{j_i} and look up the values of the gates C_{j_i} and C_{i-1}, which were computed on shorter substrings of the encoding and are now being propagated in the diagonals. To be more precise, the value of the gate C_{j_i} should be brought to the $(i-1)$th diagonal in the form of the state $x_{i-1}^{x_{j_i}}$, and then the value of C_i is computed and placed in the correct diagonal by a single transition.

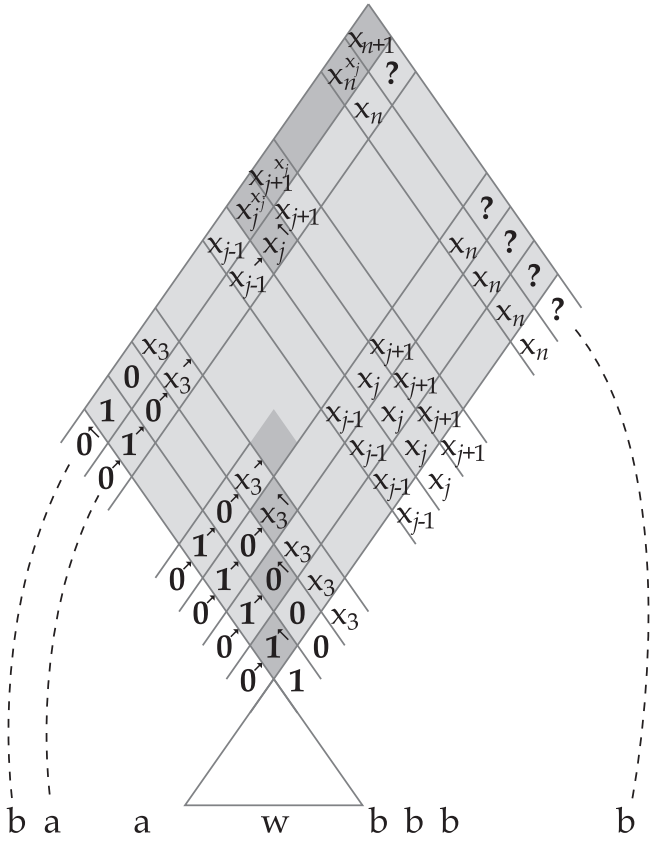


Figure 2: Computing the value of the ith gate.

The exact states of such a computation are given in Figure 2. Assume that the encoding of the $(n+1)$th gate is ba^j and it is propagated to the lower left border of Figure 2 in the form of the states $0^{\nearrow}$ for each a and the state $0^{\nwarrow}$ for b. The diagonals spawned from the b^{n+1} arrive to the left as states x_i, x_i^0 or x_i^1 for each gate i, and as ? for the last $(n+1)$-th gate. Figure 2 illustrates how the value of the $(n+1)$-th gate is computed, while the already computed values of the rest of the gates are preserved.

Furthermore, consider a full computation of the automaton on a string $ba^2ba^3ba^2babb^5 \in L_1$, given in Figure 3. This computation contains three instances of computations of the values of gates, and each case is marked with dark grey in the same way as in Figure 2.

Now it is time to define all transitions used in this computation. The

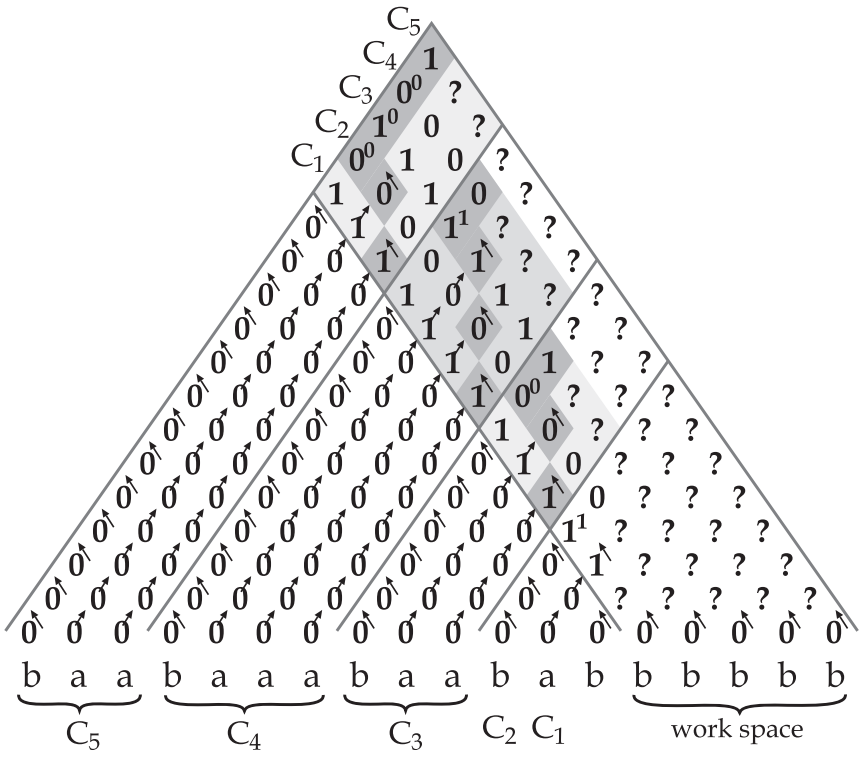


Figure 3: A sample computation of the 11-state trellis automaton.

vertical line of states in $\{0^{\nwarrow}, 1^{\nwarrow}\}$ marked with dark grey represents matching the number of as in the description of the gate to the number of diagonals with gate values, which allows seeking for the gate C_j. This vertical line is maintained by transitions of the form

$$\delta(k^{\nearrow}, \ell) = \ell^{\nwarrow} \quad (\text{for } k, \ell \in \{0, 1\}).$$

There are two cases of how this line can begin, that is, how the bottom state $1^{\nwarrow}$ is computed. If the previous gate C_n refers to a gate other than C_1, then the above general form of transitions gives $\delta(0^{\nearrow}, 1) = 1^{\nwarrow}$. However, if C_n is defined as $C_{n-1} = C_1$, then the state 1^1 will appear instead of 1 (this will be explained along with the below construction), and the following extra transition is needed to handle this case:

$$\delta(0^{\nearrow}, 1^1) = 1^{\nwarrow}.$$

The states to the left of this vertical line belong to $\{0^{\nearrow}, 1^{\nearrow}\}$, and these states are computed by the following transitions:

$$\delta(k^{\nearrow}, \ell^{\nearrow}) = \ell^{\nearrow} \quad (\text{for } k, \ell \in \{0, 1\}).$$

Beside the vertical line the transitions are:

$$\delta(k^{\nearrow}, \ell^{\nwarrow}) = \ell^{\nearrow} \quad (\text{for } k, \ell \in \{0, 1\}).$$

Now consider the states to the right of the dark grey vertical line, which are all from $\{0,1\}$. Beside the vertical line they are computed by the transitions

$$\delta(k^{\nwarrow},\ell)=\ell \quad (\text{for } k,\ell\in\{0,1\}),$$

while further to the right the transitions are

$$\delta(k,\ell)=\ell \quad (\text{for } k,\ell\in\{0,1\}).$$

All actual computations are done in the upper left border of the area in Figure 2. Assume that the gate referenced by the gate C_{n+1} is not C_1, that is, $j\geqslant 2$ (as in the figure). Then the transition in the leftmost corner of the area is

$$\delta(0^{\nwarrow},1^{\nearrow})=1,$$

(note that this place is recognized by the automaton because the value of C_1 is 1) and the border continues to the up-right by the transitions

$$\delta(k,\ell^{\nearrow})=\ell \quad (\text{for } k,\ell\in\{0,1\}).$$

Eventually the upper left border meets the dark grey vertical line, which marks the diagonal corresponding to gate C_j. The transition at this spot is

$$\delta(k,\ell)=\ell^{\ell} \quad (\text{for } k,\ell\in\{0,1\}),$$

and thus the value ℓ of the j-th gate is put to memory. This memory cell is propagated in the up-right direction by the transitions

$$\delta(k^{\ell},m^{\nwarrow})=m^{\ell} \quad (\text{for } k,\ell,m\in\{0,1\}).$$

This continues until the question mark in the $(n+1)$-th diagonal is encountered, when the value of the $(n+1)$-th gate can be computed by the following transition

$$\delta(k^{\ell},?)=\neg(k\vee\ell)\in\{0,1\} \quad (\text{for } k,\ell\in\{0,1\}).$$

Otherwise, if the $(n+1)$th gate refers to the gate C_1, then the transition in the left corner of the figure is

$$\delta(0^{\nwarrow},1^{\nwarrow})=1^{1},$$

which immediately concludes the dark grey vertical line. The rest of the computation is the same as in the above description.

Having described the contents of the upper left border of the area, it is now easy to give the transitions that compute its lower right border, as these states are computed on the basis of the upper left border of the computation for C_n. If C_n refers neither to C_1 nor to C_2, then, as shown in the figure, the second state in the lower right border is computed by the transition $\delta(1^{\nwarrow},0)=0$, which has already been defined. If C_n refers to C_1, then there

will be a state 0^1 instead of 0, and if C_n refers to C_2, there will be 0^0 in this position, so the following transitions are necessary:

$$\delta(1^{\nwarrow}, 0^k) = 0 \quad (\text{for } k \in \{0,1\}).$$

The rest of the states in the lower right border are either computed by the earlier defined transitions $\delta(k,\ell) = \ell$, or by the transitions

$$\delta(k, \ell^m) = \ell \quad (\text{for } k, \ell, m \in \{0,1\}).$$

This completes the list of transitions used to compute the value of each gate starting from C_3. A few more transitions are required to initialize the computation and to set the values of C_1 and C_2.

Each symbol b in a gate description ba^j is propagated in the right-up direction by the transition

$$\delta(0^{\nwarrow}, 0^{\nearrow}) = 0^{\nwarrow}.$$

The question marks are created from any two subsequent bs by the transition

$$\delta(0^{\nwarrow}, 0^{\nwarrow}) = ?.$$

The question marks are reduplicated by the transitions

$$\delta(q, ?) = ? \quad (\text{for } q \in \{?, 0, 1\}),$$

and by one more transition that works in the case of $C_{n+1} = C_n \downarrow C_n$:

$$\delta(0^{\nearrow}, ?) = ?.$$

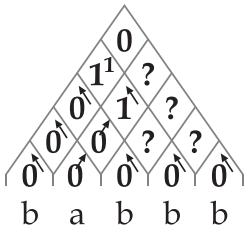


Figure 4: The beginning of the computation.

The beginning of the computation is illustrated in Figure 4: as every valid circuit description has a substring *babbb*, these transitions are needed in every computation. Here the value of C_1 is set by the transition

$$\delta(0^{\nearrow}, ?) = 1^{\nwarrow},$$

while processing the gate C_2 requires the transition

$$\delta(1^{\nwarrow}, ?) = ?.$$

This concludes the description of the transition function. To make it total, the rest of the transitions can be defined arbitrarily.

Some transitions defined above will actually never occur. Note that no sequential NOR circuit may have two consecutive gates with value 1: if $C_n = 1$, then $C_{n+1} = \neg(C_n \vee C_{j_{n+1}}) = \neg 1 = 0$. This makes the transitions $\delta(q, q')$ with $q, q' \in \{1, 1^{\nearrow}, 1^{\nwarrow}, 1^0, 1^1\}$ impossible, and as 11 such transitions have been defined above, they may be safely undefined (or redefined arbitrarily). With this correction, the transition table of the automaton is given in Table 1.

	$?$	0	1	$0^{\nearrow}$	$1^{\nearrow}$	$0^{\nwarrow}$	$1^{\nwarrow}$	0^0	0^1	1^0	1^1
$?$	$?$										
0	$?$	0	1	0	1	0^0	1^1	0	0	1	1
1	$?$	0		0		0^0		0	0		
$0^{\nearrow}$	$1^{\nwarrow}$	$0^{\nwarrow}$	$1^{\nwarrow}$	$0^{\nearrow}$	$1^{\nearrow}$	$0^{\nearrow}$	$1^{\nearrow}$				$1^{\nwarrow}$
$1^{\nearrow}$		$0^{\nwarrow}$		$0^{\nearrow}$		$0^{\nearrow}$					
$0^{\nwarrow}$	$?$	0	1	$0^{\nwarrow}$	1	$?$	1^1				
$1^{\nwarrow}$	$?$	0						0	0		
0^0	1	0^0	1^0								
0^1	0	0^1	1^1								
1^0	0	0^0									
1^1	0	0^1									

Table 1: The transition table of the 11-state trellis automaton.

The correctness of the given construction is stated in the following lemma, which specifies the state computed on (almost) every substring of a valid encoding of a circuit.

Lemma 1. *Let wb^n with $w \in \{a, b\}^*$ and $n \geqslant 2$ be a description of a circuit with the values of gates $x_1, \ldots, x_n \in \{0, 1\}$. Then:*

i. $\Delta(wb^i) \in \{x_i, x_i^0, x_i^1\}$ *for* $1 \leqslant i \leqslant n$, *and* $\Delta(wb^n) = x_n$.

ii. $\Delta(uwb^n) \in \{x_n, x_n^0, x_n^1, x_n^{\nwarrow}, x_n^{\nearrow}\}$ *for every* $u \in \{a, b\}^*$;

iii. $$\Delta(a^i w b^j) = \begin{cases} x_i^{\nearrow} & \text{if } j < i, \\ x_i^{\nwarrow} & \text{if } j = i, \\ x_i & \text{if } j > i. \end{cases} \qquad (1 \leqslant i < n,\ 1 \leqslant j \leqslant n);$$

iv. $$\Delta(ba^i w b^j) = \begin{cases} x_i & \text{if } j < i, \\ x_j^{x_i} & \text{if } j \geqslant i. \end{cases} \qquad (1 \leqslant i < n,\ 1 \leqslant j \leqslant n)$$

A formal proof is omitted, as every transition has been explained along with the construction. It could be carried out by an induction on the length of w.

This establishes the main result of this paper:

Theorem 2. *There exists an 11-state trellis automaton with 50 useful transitions that recognizes a* P*-complete language over a 2-letter alphabet.*

This automaton can be converted to a linear conjunctive grammar, which has a nonterminal representing every state and at most 4 rules for each transition.

Corollary 1. *There exists a linear conjunctive grammar with 11 nonterminals and at most 200 rules that recognizes a* P*-complete language over a 2-letter alphabet.*

Although this grammar is significantly smaller than the earlier example [8], it is still large. However, it is conjectured that the principles of the operation of this trellis automaton can be implemented in a linear conjunctive grammar much more efficiently, and a much smaller grammar generating (almost) the same language can be obtained.

References

[1] K. Culik II, J. Gruska, A. Salomaa, "Systolic trellis automata" (I, II), *International Journal of Computer Mathematics*, 15 (1984) 195–212; 16 (1984) 3–22.

[2] C. Dyer, "One-way bounded cellular automata", *Information and Control*, 44 (1980), 261–281.

[3] L. M. Goldschlager, "The monotone and planar circuit value problems are log space complete for P", *SIGACT News*, 9:2 (1977), 25–29.

[4] R. Greenlaw, H. J. Hoover, W. L. Ruzzo, *Limits to Parallel Computation: P-Completeness Theory*, Oxford University Press, 1995.

[5] O. H. Ibarra, S. M. Kim, "Characterizations and computational complexity of systolic trellis automata", *Theoretical Computer Science*, 29 (1984), 123–153.

[6] R. E. Ladner, "The circuit value problem is log space complete for P", *SIGACT News*, 7:1 (1975), 18–20.

[7] A. Okhotin, "Conjunctive grammars", *Journal of Automata, Languages and Combinatorics*, 6:4 (2001), 519–535.

[8] A. Okhotin, "The hardest linear conjunctive language", *Information Processing Letters*, 86:5 (2003), 247–253.

[9] A. Okhotin, "On the equivalence of linear conjunctive grammars to trellis automata", *Informatique Théorique et Applications*, 38:1 (2004), 69–88.

[10] A. Okhotin, "A simple P-complete problem and its representations by language equations", *Machines, Computations and Universality* (MCU 2007, Orléans, France, September 10–14, 2007), LNCS 4664, 267–278.

[11] A. R. Smith III, "Real-time language recognition by one-dimensional cellular automata", *Journal of Computer and System Sciences*, 6 (1972), 233–252.

Appendix: Tests

The trellis automaton constructed in this paper is in fact a program that solves a particular problem, and the correctness of this program has been established as a mathematical theorem. Recalling the frequently quoted words of D. E. Knuth, *"Beware of bugs in the above code; I have only proved it correct, not tried it"*, the argument would be more convincing if the correctness were to be independently verified by running the automaton on sufficiently many small inputs.

The test was organized as follows. All circuits with 2 to 10 gates of the form described in Section 4 were considered; there are $1! + 2! + 3! + \ldots + 9! = 409113$ of such circuits. Each circuit was evaluated by computing the value of each gate, and 253883 circuits were found to evaluate to 1. Independently, each circuit was represented as a string over $\{a, b\}$ according to Section 4 and the constructed 11-state trellis automaton was simulated on this input.

	value 0	value 1
Rejected	253883	0
Accepted	0	155230

These results indicate that every string given to the automaton was accepted if and only if the circuit it specifies evaluates to 1.

Turku
Centre *for*
Computer
Science

Lemminkäisenkatu 14 A, 20520 Turku, Finland | www.tucs.fi

University of Turku
- Department of Information Technology
- Department of Mathematical Sciences

Åbo Akademi University
- Department of Computer Science
- Institute for Advanced Management Systems Research

Turku School of Economics and Business Administration
- Institute of Information Systems Sciences

ISBN 978–952–12–1969–6
ISSN 1239-1891