Correction to [Rei02]

Klaus Reinhardt

Wilhelm-Schickhard Institut für Informatik, Universität Tübingen
Sand 13, D-72076 Tübingen, Germany
e-mail: reinhard@informatik.uni-tuebingen.de
April 20, 2009

Abstract. The construction in Section 13.2 in [Rei02] page 232 contains a mistake: It was not considered that the word could be too short for the variable b in the formula in the middle of the page. The section should be as follows:

13.2 Monadic second order logic

We use a method similar to the cyclically counting method in [Mat99]. In the following, we define formulas φ_n^A describing the language

$$Ł_n = 0^*10^{F(n)-1}10^*$$

with the following non-elementary function F:

$$F(1) = 1, F(n+1) = F(n)2^{F(n)}$$

This means a word $w \in \{0,1\}^*$ is in L_n iff $\varphi_n^A(w)$ is true, where A is a predicate such that for a position x in the word $A(x)$ is true iff $w_x = 1$. Obviously, any automaton recognizing the language $0^*10^{F(n)-1}10^*$ needs at least $F(n)$ states. (Otherwise, a state would appear twice between the two 1's and allow pumping the number of 0's between the two 1's.)

To do this we first define a formula ψ_n^A describing the language

$$Ł'_n = 0^*10^{F(n)-1}10^* \cup 0^*1\{0^m | m < F(n)\}$$

and then define

$$\varphi_n^A = \psi_n^A \wedge \exists x \exists y\ x \neq y \wedge A(x) \wedge A(y).$$

The formula ψ_n^A is constructed recursively over n. Let us start the inductive definition with

$$\psi_1^A = \exists x(A(x) \wedge \forall y(y = x+1 \rightarrow A(y)) \wedge \forall y(y = x \vee y = x+1 \vee \neg A(y)))$$

describing the language $0^*110^* \cup 0^*1$. The formula says that there is a position x having a value of 1, if there exists a right neighbour, it must also have 1 and all other positions have 0.

For the recursion, we use ψ_n^A to determine if two positions a and b have distance $F(n)$ or are identical in case there are less than $F(n)$ positions following.

This distance is now the length of a counter. This length is then used to control the first and last counter with InitializeC and FinalizeC respectively, and to control the correct sequence of counters with StartIncrement and Carry by locally checking all corresponding positions in neighbor counters. Recursively, we define

$$\begin{aligned}\psi^A_{n+1} = \exists x \exists y(\ & x \le y \wedge A(x) \wedge A(y) \wedge \forall z\ (z = x \vee z = y \vee \neg A(z)) \wedge \\ & \exists B \exists C\ \forall z (z < x\ \vee\ x < y < z) \rightarrow (\neg B(z) \wedge \neg C(z)) \wedge \\ & \forall a \forall b(\ (\exists A(\psi^A_n \wedge a \le b \wedge A(a) \wedge A(b) \wedge \forall z\ (z = a \vee z = b \vee \neg A(z))) \\ & \rightarrow (\text{InitializeC} \wedge \text{Block} \wedge \text{StartInc} \wedge \text{Carry} \wedge \text{FinalizeC})))\end{aligned}$$

Note that the syntax allows us to reuse the variable A, which occurs under the scope of the existential quantifier, again outside of the quantifier. This makes it possible to define φ^A_n using only a finite number of variable-symbols. Here, C contains blocks with consecutive counter representations, and B marks the beginning of each block. The recursive use of the predicate A makes sure that a and b have exactly the distance of a block length or are identical and in the last (possibly incomplete) block. This means a complete counter sequence is used to admeasure the length of only one counter for the next n.

$$\text{InitializeC} := (a = x) \rightarrow (B(a) \wedge \neg C(a) \wedge \forall c((a < c < b \vee a = b < c) \rightarrow (\neg C(c) \wedge \neg B(c)))$$

makes sure that the first block contains only zeros and that B marks exactly the beginning of the block (that is the position of the least significant bit).

$$\text{Block} := (x \le a < b \le y\ \vee\ x = y \le a < b) \rightarrow (B(a) \leftrightarrow B(b))$$

takes care that B is continued which means it has a 1 exactly at the beginning of the each block within the area where the counters have to work.

$$\text{StartInc} := (B(a) \wedge a \neq b) \rightarrow (\neg(C(a) \leftrightarrow C(b)))$$

makes sure that the first (least significant) bit of the counter in each block changes each time.

$$\text{Carry} := ((C(a) \wedge \neg C(b)) \leftrightarrow (\neg C(a+1) \leftrightarrow C(b+1))) \vee B(a+1) \vee a = b$$

makes sure that a 1 changes to a 0 exactly if, in the corresponding bit in the following block, the next bit (if it was not the last in the block) must change.

$$\begin{aligned}\text{FinalizeC} := & B(y) \wedge \\ & (x \neq y \wedge a \neq b) \rightarrow ((b = y) \leftrightarrow (B(a) \wedge \forall c(a \le c < b \rightarrow C(c)))) \wedge \\ & (a \neq b \wedge B(a) \wedge \forall c(a \le c < b \rightarrow C(c))) \rightarrow (b = y)\end{aligned}$$

makes sure that the last block is the one containing only 1's in the counter and this final counter requires the y to be there.

Theorem 1. 13.1 *The formula φ^A_n defined above has size $O(n)$ and defines the language $\{0^*10^{F(n)-1}10^*\}$ for which a finite automaton needs at least $F(n)$ states.*

Example:
The language $0^*10^{2047}10^*$ is described by φ_4^A. Here, the existentially quantified C contains all binary representations of numbers from 0 to 255 having length 8. To check the correctness of C and the block-marks in B, the formula recursively uses φ_3^A describing $0^*10^710^*$. In this description by φ_3^A, the corresponding C contains all binary representations of numbers from 0 to 3 having length 2. This recursively uses φ_2^A describing 0^*1010^*; the corresponding C contains just 0 and 1 finally using φ_1^A.

A: 0...0100000000000000000000000... 000000000000000010...
C: 000000001000000001000000... 011111111111111100...
B: 100000001000000010000000... 100000001000000010...
A: 0...01000000010...
C: 0010011100...
B: 1010101010...
A: 0...01010...
C: 0100...
B: 1110...

Acknowledgment: I thank Dietrich Kuske for pointing out the mistake.

References

[Mat99] O. Matz. *Dot-Depth and Monadic Quantifier Alternation over Pictures.* Technical report, Aachener Informatik Berichte 99-08, RWTH Aachen, 1999.

[Rei02] K. Reinhardt. The complexity of translating logic to finite automata. In Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors, *Automata, Languages, and Infinite Games*, volume 2500 of *Lecture Notes in Computer Science*, pages 231–239. Springer, 2002.