Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT
Theoretical Computer Science 313 (2004) 447–462

Theoretical Computer Science

www.elsevier.com/locate/tcs

Assembling molecules in ATOMIX is hard

Markus Holzer[a,*], Stefan Schwoon[b]

[a] *Institut für Informatik, Technische Universität München, Arcisstraße 21, München D-80290, Germany*
[b] *Institut für Formele Methoden der Informatik, Universität Stuttgart, Universität sstr. 38, 70569 Stuttgart, Germany*

Received 30 April 2002; received in revised form 16 September 2002; accepted 4 November 2002

Abstract

It is shown that assembling molecules in the ATOMIX game can be used to simulate finite automata. In particular, an instance of ATOMIX is constructed that has a solution if and only if the non-emptiness intersection problem for finite automata is solvable. This shows that the game under consideration is PSPACE-complete, improving a recent result of Hüffner et al. (Lecture Notes in Computer Science, Vol. 2174, Springer, Vienna, Austria, 2001, pp. 229–243). Moreover, the given reduction shows that there are ATOMIX games which have exponentially long optimal solutions. We also give an easy construction of ATOMIX game levels whose optimal solutions meet the worst case.

Keywords: Finite automata, Intersection emptiness, Block sliding puzzle; PSPACE-completeness

1. Introduction

ATOMIX is a solitaire game invented by Günter Krämer in 1990 and first published by Thalion Software. The game takes place on a rectangular finite two-dimensional grid, the board. Every cell of the board is either a wall, contains an atom, or is free. Walls cannot be changed, and atoms can be of different types. A move consists of pushing an atom along the x-axis or the y-axis. When an atom is pushed, it continues moving until it reaches a wall or another atom. The game is won when all atoms are arranged in a given "molecule" goal pattern. An instance of an ATOMIX game assembling the water molecule is depicted in Fig. 1—walls are represented by black squares,

* Corresponding author.
E-mail addresses: holzer@informatik.tu-muenchen.de (M. Holzer), schwoosn@fmi.uni-stuttgart.de (S. Schwoon).

0304-3975/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2002.11.002

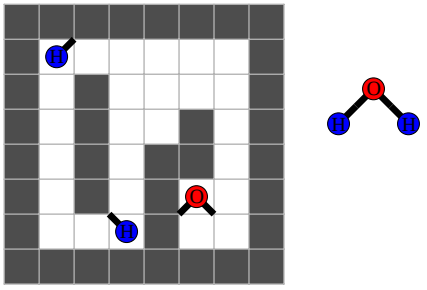


Fig. 1. An ATOMIX problem assembling the water molecule, (a) ATOMIX board, (b) molecule.

free space by white squares, and atoms by labelled circles with connections. A non-optimal sample solution within 16 moves of the ATOMIX puzzle from Fig. 1 is shown in Fig. 2. We encourage the reader to find the optimal 11 moves solution. Implementations of certain ATOMIX variations are available on the Internet; e.g., for the X Window System a version under the terms of the General Public License (GPL) can be downloaded from `www.informatik.uni-oldenburg.de/~pearl/gnome/atomix.html`. It is worth mentioning that this program also contains a level editor. A JavaScript version can be played online at `www.sect.mce.hw.ac.uk/~peteri/atomix/`.

Formally the ATOMIX problem is defined as follows: given an ATOMIX board and a molecule, is there a sequence of moves to assemble the atoms on the board to form the given molecule? Obviously, this problem can be formalised as a state space search problem, which recently was done by Hüffner et al. [8]. There, different heuristic search methods were presented. ATOMIX falls into the category of sliding block puzzles as, e.g., PushPush [3], Sokoban [2,4], or 15-Puzzle [10], where time and space complexity was, and still is, subject of intense research. Though seemingly trivial, most variations are at least NP-hard, and contained in PSPACE; some are even PSPACE-complete—we refer the reader to, e.g., Balcázar et al. [1] for further details on computational complexity. Hüffner et al. [8] actually have shown that ATOMIX is NP-hard and contained in PSPACE, while the exact complexity was stated as an open problem. In this paper we solve this open problem and improve their result showing the following theorem.

Theorem 1. ATOMIX *on an* $n \times n$ *board is PSPACE-complete.*

To this end we show that ATOMIX game puzzles can simulate deterministic or non-deterministic finite automata. In particular, we construct an ATOMIX instance that is solvable if and only if the non-emptiness intersection problem for finite automata, a problem known to be PSPACE-complete [6,9], has a solution. Observe that most importantly the proof of the above given theorem will show that there are ATOMIX instances which have exponentially long optimal solutions.

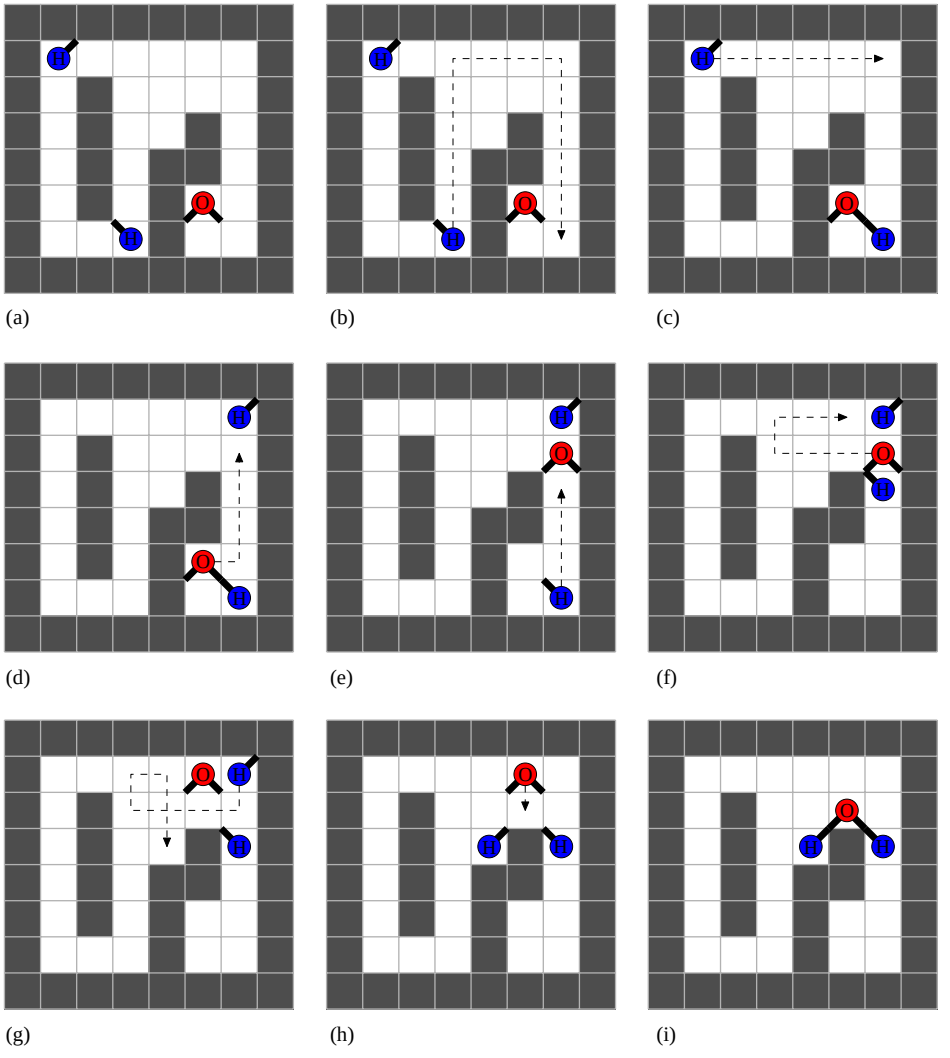


Fig. 2. A sample solution within 16 moves, (a) initial config., (b) 1st–3rd move, (c) 4th move, (d) 5th–6th move, (e) 7th move, (f) 8th–10th move, (g) 11th–15th move, (h) 16th move, (i) solution.

The paper is organised as follows: In Section 2 we introduce the basic building blocks of our solution. Then in Section 3 we present how ATOMIX puzzles can simulate non-deterministic finite automata and how synchronisation between finite automata can be achieved. Section 4 shows how to implement an n-bit counter using ATOMIX. These puzzles have exponentially long optimal solutions. Finally, Section 5 contains our conclusions and pointers to related papers.

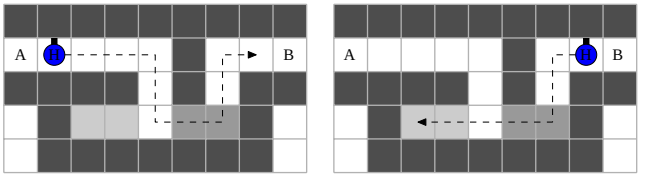


Fig. 3. One-way device.

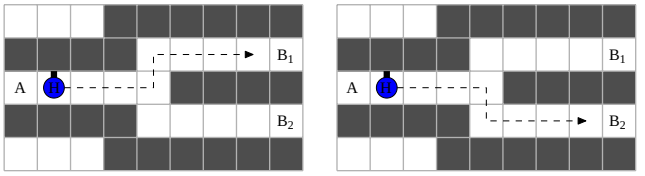


Fig. 4. Non-deterministic choice.

2. Basic devices

The construction given in this paper relies on a number of basic devices with special properties. In the figures below, walls are represented by dark squares and free spaces by white squares. Occasionally, free spaces will be shaded to indicate areas of special importance. Atoms are represented by labelled circles with certain connections; the labelling has no special significance.

2.1. One-way device

The first device is the *one-way box* shown in Fig. 3. Its purpose is to allow the passage of atoms from the entry A to the exit B but not vice versa. Both of the shaded areas in the figure should be extended until they are large enough to hold all the atoms in the puzzle.

Any atom can enter the device at A and leave at B. Atoms entering the device at B will end up in the "blind alley," the light shaded area. If the blind alley is large enough, the atoms can never fill it up completely, and thus no atom is able to go back to A. Similarly, when atoms move back from the alley, they must not be able to fill up the dark shaded area. Note, that in our constructions later on it will often be enough to use a blind alley of size one or two.

2.2. Choice and merge

The second device is the *choice device* whose inner part is depicted in Fig. 4. An atom can enter at A and exit at either B_1 or B_2; additional one-way devices at each gate ensure entry only at A and exit only at B_1 or B_2. Multiple devices of this type

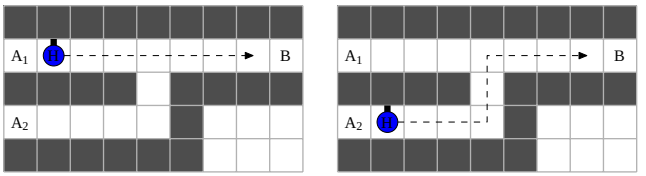


Fig. 5. Merge device.

can be used in sequence to allow a set of choices S; we call such a construction a choice device over S.

The *merge device* in Fig. 5 implements the reverse; atoms can enter at either A_1 or A_2 and exit at B; again we assume appropriate one-way devices at the entries and the exits. The merge device can be extended to accommodate more than two entries.

2.3. Catalyst chamber

The fourth device is slightly more complicated. Consider the *catalyst chamber* shown in Fig. 6(a). Again, there are one-way devices at the entries and exits to ensure that entry is only possible at A, and that exit is only possible at B_1 and B_2. The device has the following properties: [1]

(1) If one atom enters the catalyst chamber alone, it can never exit,
(2) if two atoms enter the catalyst chamber, both of them can leave the chamber through different exits, and
(3) if two atoms enter the catalyst chamber, they cannot leave through the same exit.

Property 1 is easy to verify; once an atom enters the chamber alone, it can only move from corner to corner but never enter the exit corridors. Property 2 is shown by the example sequence shown in Fig. 6, parts (b)–(d). Observe, that the device does not guarantee that a particular atom leaves through a particular exit. However, this is not important for our constructions. We prove Property 3 by showing that the following property is an invariant, provided that there are only two atoms in the chamber, both of which enter at A:

(3′) The shaded areas in Figs. 6(e) and (f) never hold two atoms simultaneously.

Property 3 then immediately follows from 3′. The Property 3′ holds when the atoms first enter the chamber. Moreover, from any situation in which both atoms are outside a shaded area, we can bring at most one of them into the area with one move. Assume therefore that there is already one atom inside one of the shaded areas, and the other outside of it. However, there is no such configuration that would allow the outside atom to be moved into the area; the reader can easily verify this by trying all the possibilities.

[1] We have constructed a finite automaton simulating the behaviour of the catalyst chamber. This has resulted in a machine with 324 states (177 of them were reachable) and 1570 transitions. Finally, we have verified the stated properties using the symbolic computation environment Grail+ [11]. After some inspection of the finite state automaton we have found a simple proof for Property 3, which is given below.

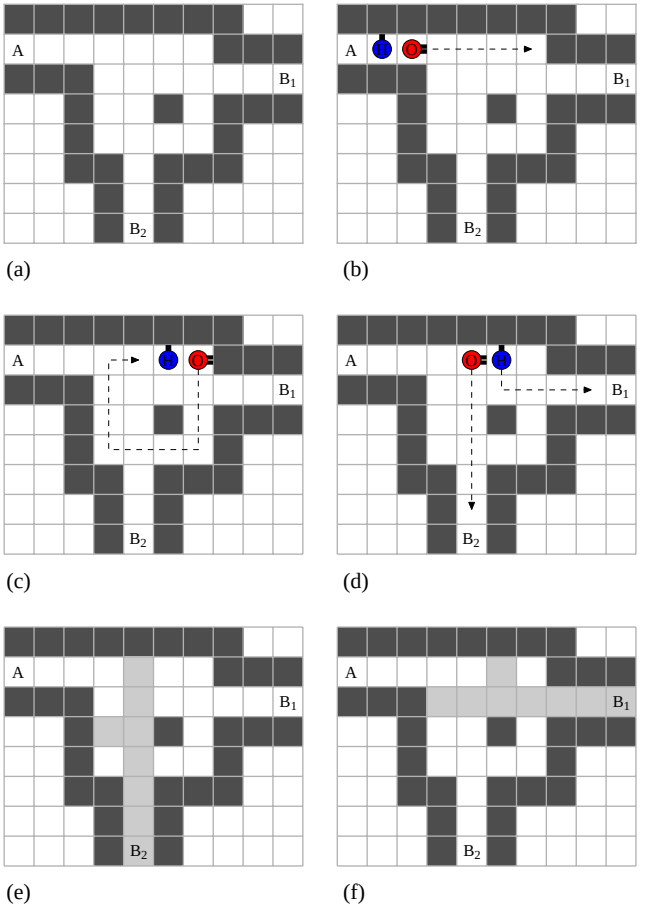


Fig. 6. Catalyst chamber, (a) catalyst chamber, (b) Property 2: 1st and 2nd move, (c) Property 2: 3rd–6th move, (d) Property 2: 7th–9th move, (e) Property 3′: 1st region, (f) Property 3′: 2nd region.

2.4. Wires and planar crossings

To connect the devices presented previously, we also need to provide "wires" and a way to cross wires in two dimensions. Fortunately, both are trivial to implement; a wire is constructed by a line of free spaces flanked by walls on both sides; bent wires are also possible. Crossing wires is equally simple. Both devices are shown in Fig. 7.

The property of the wire is simply that an atom can pass from A to B and vice versa. The property of the crossing is that if an atom enters at A_1 it can only exit at B_1 and that entry at A_2 forces exit at B_2 and vice versa. In our constructions later on this property will be ensured by the fact that both the horizontal and the vertical wire can never hold more than one atom at any given time. Therefore no "arm" of the device can be filled up and allow illegal passages. Another (simple) way of preventing filling the wires is making them long enough.

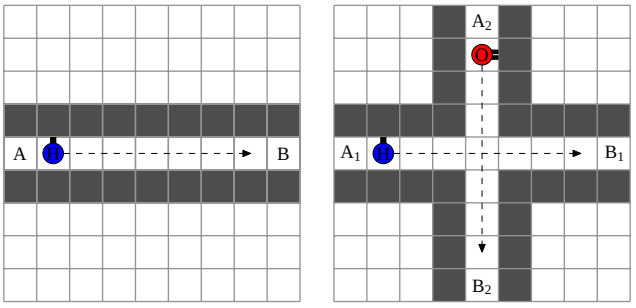


Fig. 7. Two simple devices, (a) wire, (b) crossing.

3. ATOMIX is PSPACE-complete

In this section we show that ATOMIX on an $n \times n$ board is PSPACE-complete. Obviously, the problem is contained in PSPACE. For the convenience of the reader we recall the simple argument given by Hüffner et al. [8]. Given an instance of ATOMIX a non-deterministic Turing machine can solve it by repeatedly applying a legal move until a goal is reached. The number of possible ATOMIX board configurations is limited by $n^2!$; hence the machine can announce that the game is unsolvable after having applied more moves without finding a solution. Since an encoding of an ATOMIX configuration needs polynomial space, it follows that ATOMIX is in NPSPACE = PSPACE.

For the hardness we argue as follows. First we show how an ATOMIX puzzle can simulate the behaviour of a finite automaton. The puzzle is constructed in such a way that the puzzle has a solution exactly if the language of the automaton is not empty. In the second part of the proof we extend the construction such that synchronisation on an input symbol can be simulated. Given a sequence of finite automata, we can then create a puzzle that has a solution exactly if there is a word which is accepted by all automata. The constructed puzzle is polynomial in the size of the automata. Since by Galil [6] and Kozen [9] it is PSPACE-complete to check if the intersection of an arbitrary number of regular languages, given by deterministic or nondeterministic finite automata, is non-empty, this proves PSPACE-hardness of the solvability question for ATOMIX puzzles.

To simplify the presentation we use diagrams for the basic devices presented in Section 2. The diagrams are shown in Fig. 8. Throughout the diagrams, we use simple lines for wires.

Fig. 8(a) represents a *choice device* with one entry and one exit for each element of the set S; Fig. 8(b) is a generalised form where $|R|$ multi-choice devices over S are placed in parallel such that an atom entering at a gate labelled $r \in R$ can exit at some gate labelled $(r,s) \in R \times S$. Figs. 8(c) and (d) are the analogous *merge devices*; Fig. 8(c) has one entry for each element of the set S, and Fig. 8(d) has one entry for each tuple in $R \times S$ and one exit for each element of R. An atom entering

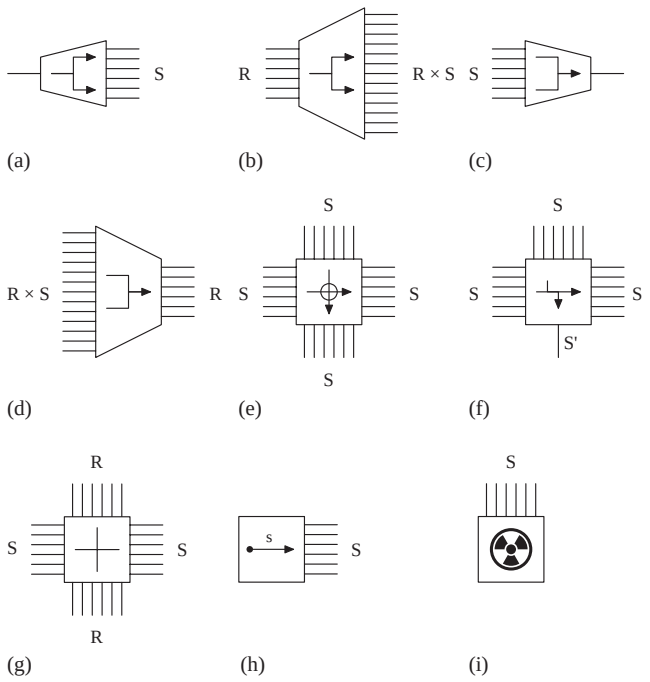


Fig. 8. Basic diagrams: (a) choice device, (b) generalised choice, (c) merge device, (d) generalised merge, (e) synchronisation box, (f) filter, (g) crossing box, (h) initialisation box, and (i) reaction chamber.

Fig. 8(d) at the gate (r,s) exits at the gate labelled r. The *synchronisation box* in Fig. 8(e) essentially consists of $|S|$ catalyst chambers, one for each element of S; there are two sets of entries (left and upper side) and two sets of exits (right and lower side) for all elements of S. The entries and exits for $s \in S$ connect to the catalyst chamber for s. Two atoms entering the box on the left and at the top can leave the box only if they enter at the same element of S. Fig. 8(f) has the function of a *filter*. Entries are labelled with elements of the set S; for convenience, we provide two entries for each element at the top and left of the box. Exits (on the right) are also labelled with S, except for a special exit at the bottom. An atom entering the box at either entry $s \in S'$, where S' is some subset of S, can choose to leave at the exit s or at the special exit. An atom entering at $s \in S \backslash S'$ can only leave at the normal exit s. Such a filter can easily be built with the standard devices presented before; we omit the details. The *crossing box* is shown in Fig. 8(g) and it allows atoms to pass through without any interaction, i.e., an atom entering at $s \in S$ ($r \in R$, respectively) leaves the gate at exit s (r, respectively). Next, Fig. 8(h) shows the *initialisation box*. It contains one oxygen atom as shown in Fig. 9(a) and one exit for each element of the set S; the atom can leave the box only at the exit labelled s, for some $s \in S$. If S is a singleton set, the box simply designates the position of one atom in the initial configuration. Finally, Fig. 8(i) depicts the *reaction chamber*. It contains two hydrogen atoms as shown in

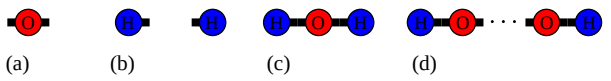


Fig. 9. Atoms and molecules, (a) oxygen, (b) hydrogen, (c) water, (d) H_2O_n molecule.

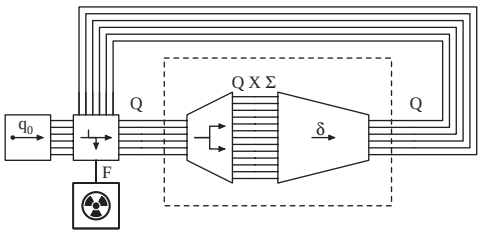


Fig. 10. Symbolic diagram of an automaton.

Fig. 9(b) and entries for each element of the set S; atoms in the box cannot leave since the entries are protected by appropriate one-way devices and are used to form the molecule goal pattern. Hence, atoms are forced to enter the reaction chamber in order to assemble the goal molecule.

3.1. Simulating a finite automaton

Fix a finite automaton $A = (Q, \Sigma, \delta, q_0, F)$ for the rest of the section where Q is a set of states, Σ the input alphabet, $\delta : Q \times \Sigma \to 2^Q$ the (non-deterministic) transition function, q_0 the initial state, and F the set of final states. We shall construct an ATOMIX puzzle that has a solution exactly if the language of A is non-empty, i.e., $L(A) \neq \emptyset$. Almost all of the "computation" will be carried out by a single atom. The puzzle can be solved if the atom can enter the reaction chamber in which it can combine with other atoms to form the goal molecule; this is the water molecule shown in Fig. 9(c).

The main part of the construction consists of the simulation of δ, i.e., a device that, given a tuple $(q, a) \in Q \times \Sigma$, yields a successor state $q' \in \delta(q, a)$; see Fig. 10. In ATOMIX terms, an atom can enter at a gate labelled by (q, a) and exit at a gate labelled by q'. Each entry gate in the device is connected to a choice device over $\delta(q, a)$. All exits from the choice devices that belong to the same successor state are then merged at the respective exit. In the following we shall represent such a transition device by a box labelled δ.

The complete puzzle for the automaton is symbolically shown in Fig. 10. The dashed box has $|Q|$ entries on the left indicating the state of the automaton before reading an input symbol. An atom traversing the box first selects an input symbol, then simulates the transition function to find the successor state. Each pass of the atom through the box simulates the processing of one input symbol chosen arbitrarily by the atom. Moreover, if the atom is in an entry belonging to a final state, the filter enables it to stop reading input and enter the reaction chamber. The puzzle is completed by connecting each exit

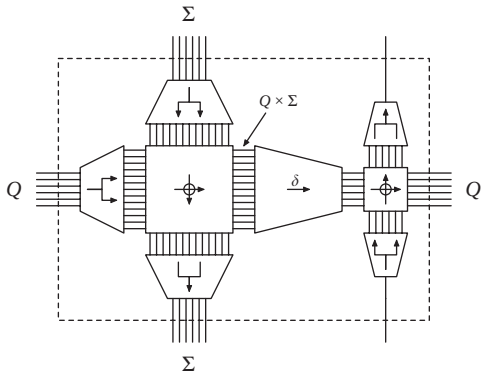


Fig. 11. Interaction between synchronisation process and automaton.

of the box to the corresponding entry of the filter and placing an atom at the entry belonging to the initial state. The puzzle can be solved if the atom is able to reach the reaction chamber, i.e., if there is a word accepted by the automaton.

3.2. Simulating an arbitrary number of finite automata

Now that we know how to simulate one deterministic or non-deterministic finite automaton we can solve the main problem. Given a sequence $A_1, \ldots, A_n$ of finite automata, with $A_i = (Q_i, \Sigma, \delta_i, q_i, F_i)$, all using a common alphabet Σ, the non-emptiness intersection problem consists of checking whether the intersection of the languages of all n automata is non-empty, i.e., $\bigcap_{i=1}^{n} L(A_i) \neq \emptyset$. The basic idea of the construction is to create puzzles for each automaton first, and to connect them all to a common reaction chamber such that the puzzle can be solved if each computation atom can enter the chamber. However, we are only interested in solutions where all automata accept the same input, therefore additional machinery is needed.

We create a so-called synchronisation process inhabited by a single atom, the synchronisation atom. In each step of the computation, this atom either chooses to terminate or selects the next input symbol to be processed. It then interacts with the puzzles created from the automata and makes sure that all of them have made the same choice, otherwise the puzzle will be left in an unsolvable state. The interaction of the synchronisation process with a single automaton takes place in two phases and is shown in Fig. 11. The dashed box shows the processing of a single input symbol; again, the box has $|Q|$ entries (left) and exits (right) to indicate the state the automaton is in. The other control structures of the automaton have been omitted for clarity.

In the first phase, the synchronisation atom enters at one of the $|\Sigma|$ entry gates at the top left and leaves through the corresponding gate on the bottom left. The computation atom of the automaton first chooses an input symbol, trying to "guess" the same symbol as the synchronisation atom, whereas the synchronisation atom guesses the state that the computation atom is in. If both guesses are right, then both atoms can pass through

the synchronisation box in the left of the figure. The computation atom can then pass through the transition function while the synchronisation atom goes on to perform the first phase with other automata. The second phase takes place after the synchronisation process has interacted with all automata and made sure that all of them selected the same input symbol. The synchronisation atom now starts a second pass through all automata, entering Fig. 11 on the bottom right and guessing the successor state of the automaton. If the choice was right, it passes through the synchronisation box on the right, leaving at the gate in the top right of the figure. This step makes sure that the computation atom of the automaton really simulates the transition function; without this step the computation atom would be allowed not to leave the first synchronisation box and wait for the synchronisation atom to come back in the next round, thus skipping one input symbol.

An overview of the synchronisation process is given in Fig. 12. Every traversal of the synchronisation process corresponds to reading one input symbol. The synchronisation atom starts at the top of the figure, selects an input symbol, then interacts with the automata in two passes and returns to the beginning. In each step, the synchronisation atom is also allowed to enter the reaction chamber instead of "reading" a symbol. Similarly, the atoms within the automata are allowed to enter the chamber if they are in a final state before reading a symbol. All entries to the reaction chamber are protected by one-way devices. Therefore, the (artificial) goal molecule[2] H_2O_{n+1}, which is of a very simple structure and shown in Fig. 9(d), can be assembled if and only if all automata reach a final state at the same time. More precisely, this happens after all automata have processed the same input, which accomplishes the reduction to the non-emptiness intersection problem.

Note that the construction relies on atoms making the right "guesses" during the synchronisation process. If any of these guesses is wrong, the puzzle will be left in an irrecoverable state. Similarly, all atoms have to terminate the computation at the same time. However, we can ignore wrong guesses since we are only interested in the mere existence of a solution.

4. On exponentially long optimal solutions

From the proof of the main result in the previous section we conclude that there exist ATOMIX game instances which have superpolynomially, or to be more precise, exponentially long optimal solutions. Although most constructions we used are much like those in the popular game, it is a tedious exercise to construct a particular ATOMIX instance with this property based on the construction of the previous section. In this section, we give an easier construction for puzzles with exponentially long optimal solutions. To this end we realize a pseudo n-bit counter in ATOMIX.

The main part of the construction consists of the simulation of a single bit, i.e., a device that stores a bit, changes it, and produces a carry bit if triggered by an

[2] Instead of using hydrogen and oxygen one might use any other atoms as, e.g., uranium, which might be dangerous in case it exceeds the critical mass. Do not try this at home.

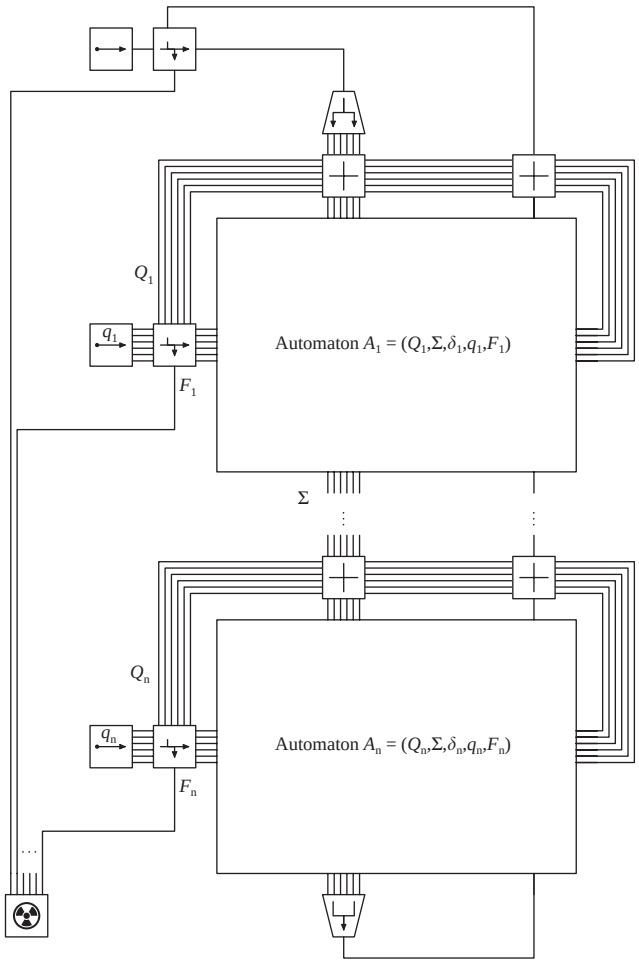


Fig. 12. Overview of synchronisation process.

increment event. Before we describe the bit device in more detail, we must give some further details on the counting process. In order to obtain an easy device, i.e., with a small number of basic gates, we implement a pseudo-counting process. This means that the stored 0 bit can, but need not, be changed to 1 and produces no-carry by an increment event, while a 1 bit must be changed to 0 and produces a carry bit. This slight difference to ordinary counting will allow us to save some gates. The complete puzzle for the *bit device* is symbolically shown in Fig. 13. In ATOMIX terms, a single atom trapped in one of two catalyst chambers can store binary information; we call this atom the *storage atom*. Another atom, the *increment atom*, can enter at gate labelled by *incr* and exit at a gate labelled *carry* or *no-carry* depending on the element the device has stored. Moreover, there is also the possibility that the storage atom may leave the device at exit labelled *clear* and enter the reaction chamber.

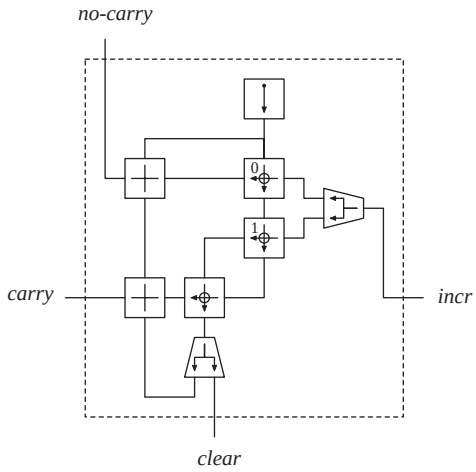


Fig. 13. Symbolic diagram of a bit device.

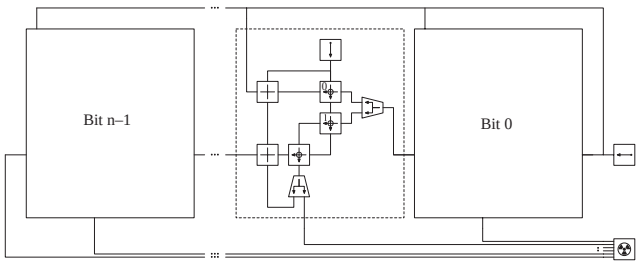
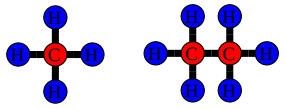


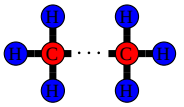
Fig. 14. Overview of a pseudo-n-bit counter.

The bit device is initialised by moving the storage atom to the catalyst chamber labelled 0. An increment atom enters at entry *incr* at the right and first chooses a bit, trying to "guess" the bit stored by the storage atom in the device. If the guess is right, then both atoms *can* pass through the appropriate catalyst chamber. In case both atoms have met in catalyst chamber labelled 0 one atom may leave through the gate labelled *no-carry* on top. To ensure that both atoms leave the catalyst chamber labelled 1 an "after-burn" catalyst chamber is needed. Then one atom can leave through the *carry* gate on the left while the other one can choose between catalyst chamber labelled 0 or the *clear* gate on the bottom. Observe that the missing after-burn catalyst chamber for the sub-device labelled 0 is the reason that the constructed device implements a pseudo-bit counter.

An overview of the pseudo n-bit counter is given in Fig. 14. The increment atom starts at the right of the figure triggering an increment operation on the n-bit number. The atom traverses the bit devices in sequence until it exits at a gate labelled *no-carry*, and returns to the beginning. Each storage atom is only allowed to enter the reaction



(a) Methane. (b) Ethane.



(c) C_nH_{2n+2} alkane.

Fig. 15. More natural molecules.

chamber via the clear gate; this is only allowed if the bit value flips from 1 to 0. The only way the increment atom may reach the reaction chamber is through the carry gate from the $(n-1)$th bit position (highest bit). All entries to the reaction chamber are protected by one-way devices. Therefore, the artificial goal molecule H_2O_{n+1} shown in Fig. 9(d), can be assembled if and only if one counts from 0 up to 2^n. Thus, the optimal solution to the puzzle is of exponential length.

5. Conclusion

In this paper we have shown that the Atomix game is PSPACE-complete by reducing the non-emptiness intersection problem for deterministic or non-deterministic finite automata to the problem under consideration. This solves an open problem stated by Hüffner et al. [8], and the reduction proves that there exist Atomix game instances which have exponentially long optimal solutions. Moreover, we have also presented an easy construction to design quite complicated Atomix game levels.

It is worth to mention, that from the proof of the main result on the PSPACE-completeness one concludes that the atom type as well as the form of the goal molecule is not important for the construction. Hence, the computational complexity of Atomix stems from the labyrinth structure. Nevertheless, a point of critics on our result may be the usage of the artificial goal molecules H_2O_n. It is obvious, to alter the given construction, that it works for more *natural* goal molecules as, e.g., the alkanes. The alkanes—paraffins—are the simplest homologous series of organic compounds of hydrogen and carbon, where all atoms are linked by single bonds. The alkane with the simplest structure is methane, CH_4. It consists of a central carbon atom, surrounded by four hydrogen atoms and is shown in Fig. 15(a).

Each succeeding member of the alkane series has a further methylene group, $–CH_2–$, in the chain; ethane follows methane and is depicted in Fig. 15(b). More generally, the formula for the alkane series of hydrocarbons is C_nH_{2n+2} and the corresponding molecule is depicted in Fig. 15(c).

Finally, in Table 1 we summarise some complexity results on block sliding puzzles, into which Atomix falls. As already mentioned in the introduction most of these

Table 1
Computational complexity of some block sliding puzzles summarised

Game	1. Robot	2. Pull	3. Blocks	4. Fix.	5. No.	6. Path	7. Slide	8. Dim.	9. Complexity
PushPush3D	+	−	Unit	−	1	+	+	3D	NP-hard [12]
PushPush	+	−	Unit	−	1	+	+	2D	NP-hard [3]
Push-*	+	−	Unit	−	k	−	−	2D	NP-hard [7]
Sokoban+	+	−	1×2	+	2	−	−	2D	PSPACE-compl. [4]
Sokoban	+	−	Unit	+	1	−	−	2D	PSPACE-compl. [2]
15-Puzzle	−		Unit	−	1	−	−	2D	NP-hard [10]
ATOMIX	−		Unit	+	1	−	+	2D	PSPACE-compl.
RushHour	−		$1 \times \{2,3\}$	−	1	+	−	2D	PSPACE-compl. [5]

problems are NP-hard and contained in PSPACE; some of them are even PSPACE-complete. The following table is taken from Demaine et al. [3], extended by the category of games where blocks are pushed by an external agent not presented on the board. The columns mean:

(1) Are the moves done by a robot on the board, or by an outside agent?
(2) Can the robot pull as well as push?
(3) Are all blocks unit squares, or may they have different shapes?
(4) Are there fixed blocks, or are all blocks movable?
(5) How many blocks can by pushed at a time?
(6) Does it suffice to move the robot/a special block from s to t, instead of pushing all blocks into storage location?
(7) Will the blocks "keep sliding" when pushed till they hit an obstacle?
(8) The dimension of the puzzle: is it 2D or 3D?
(9) Computational complexity: hardness or completeness?

Acknowledgements

Thanks to Peter Rossmanith and Falk Hüffner for fruitful discussions and to the anonymous referees for their valuable comments on the subject.

References

[1] J.L. Balcázar, J. Díaz, J. Gabarró, Structural Complexity I, in: EATCS Monographs on Theoretical Computer Science, Vol. 11, Springer, Berlin, 1988.
[2] J. Culberson, Sokoban is PSPACE-complete, in: Internat. Conf. on Fun with Algorithms, Proc. in Informatics 4, Carleton Scientific, Waterloo, Canada, Elba, Italy, 1999, pp. 65–76.
[3] E.D. Demaine, M.L. Demaine, J. O'Rourke, PushPush and Push-1 are NP-hard in 2D, in: Proc. 12th Annual Canadian Conf. on Computational Geometry, Fredericton, New Brunswick, Canada, 2000, pp. 17–20.
[4] D. Dor, U. Zwick, Sokoban and other motion planning problems, Comput. Geom. Theory Appl. 13 (4) (1999) 215–228.

[5] G.W. Flake, E.B. Baum, RushHour is PSPACE-complete, or Why you should generously tip parking lot attendants, Theoret. Comput. Sci. 270 (1–2) (2002) 895–911.
[6] Z. Galil, Hierarchies of complete problems, Acta Inform. 6 (1976) 77–88.
[7] M. Hoffmann, Motion planning amidst movable square blocks: Push-* is NP-hard, in: Proc. 12th Ann. Canadian Conf. on Computational Geometry, Fredericton, New Brunswick, Canada, 2000, pp. 205–209.
[8] F. Hüffner, S. Edelkamp, H. Fernau, R. Niedermeier, Finding optimal solutions to Atomix, in: Proc. Joint German/Austrian Conf. on AI: Advances in Artificial Intelligence, Lecture Notes in Computer Science, Vol. 2174, Springer, Vienna, Austria, 2001, pp. 229–243.
[9] D. Kozen, Lower bounds for natural proof systems, in: Proc. 18th Annual Symp. on Foundations of Computer Science, 1977, pp. 254–266.
[10] D. Ratner, M.K. Warmuth, The $(n^2 - 1)$-puzzle and related relocation problems, J. Symbolic Comput. 10 (2) (1990) 111–137.
[11] D. Raymond, D. Wood, Grail: A C++ library for automata and expressions, J. Symbolic Comput. 11 (1995) 1–10.
[12] J. O'Rourke, the Smith Problem Solving Group, PushPush is NP-hard in 3D, Technical Report 064, Smith College, Northampton, MA, November 1999.