Volume 18 Number 1



# ACTA CYBERNETICA



Szeged, 2007

# ACTA CYBERNETICA

**Information for authors.** Acta Cybernetica publishes only original papers in the field of Computer Science. Manuscripts must be written in good English. Contributions are accepted for review with the understanding that the same work has not been published elsewhere. Papers previously published in conference proceedings, digests, preprints are eligible for consideration provided that the author informs the Editor at the time of submission and that the papers have undergone substantial revision. If authors have used their own previously published material as a basis for a new submission, they are required to cite the previous work(s) and very clearly indicate how the new submission offers substantively novel or different contributions beyond those of the previously published work(s). Each submission is peer-reviewed by at least two referees. The length of the review process depends on many factors such as the availability of an Editor and the time it takes to locate qualified reviewers. Usually, a review process takes 6 months to be completed. There are no page charges. Fifty reprints are supplied for each article published.

**Manuscript Formatting Requirements.** All submissions must include a title page with the following elements:

- title of the paper
- author name(s) and affiliation
- name, address and email of the corresponding author
- An abstract clearly stating the nature and significance of the paper. Abstracts must not include mathematical expressions or bibliographic references.

References should appear in a separate bibliography at the end of the paper, with items in alphabetical order referred to by numerals in square brackets. Please prepare your submission as one single PostScript or PDF file including all elements of the manuscript (title page, main text, illustrations, bibliography, etc.). Manuscripts must be submitted by email as a single attachment to either the most competent Editor, the Managing Editor, or the Editor-in-Chief. In addition, your email has to contain the information appearing on the title page as plain ASCII text. When your paper is accepted for publication, you will be asked to send the complete electronic version of your manuscript to the Managing Editor. For technical reasons we can only accept files in LaTeX format.

**Subscription Information.** Acta Cybernetica is published by the Institute of Informatics, University of Szeged, Hungary. Each volume consists of four issues, two issues are published in a calendar year. Subscription rates for one issue are as follows: 5000 Ft within Hungary, €40 outside Hungary. Special rates for distributors and bulk orders are available upon request from the publisher. Printed issues are delivered by surface mail in Europe, and by air mail to overseas countries. Claims for missing issues are accepted within six months from the publication date. Please address all requests to:

Acta Cybernetica, Institute of Informatics, University of Szeged
P.O. Box 652, H-6701 Szeged, Hungary
Tel: +36 62 546 396, Fax: +36 62 546 397, Email: acta@inf.u-szeged.hu

**Web access.** The above informations along with the contents of past issues are available at the Acta Cybernetica homepage http://www.inf.u-szeged.hu/actacybernetica/ .

**Zoltán Ésik**
Department of Foundations of
Computer Science
University of Szeged
Szeged, Hungary
ze@inf.u-szeged.hu

**Zoltán Fülöp**
Department of Foundations of
Computer Science
University of Szeged
Szeged, Hungary
fulop@inf.u-szeged.hu

**Ferenc Gécseg**
Department of Computer Algorithms
and Artificial Intelligence
University of Szeged
Szeged, Hungary
gecseg@inf.u-szeged.hu

**Jozef Gruska**
Institute of Informatics/Mathematics
Slovak Academy of Science
Bratislava, Slovakia
gruska@savba.sk

**Helmut Jürgensen**
Department of Computer Science
Middlesex College
The University of Western Ontario
London, Canada
helmut@csd.uwo.ca

**Alice Kelemenová**
Institute of Computer Science
Silesian University at Opava
Opava, Czech Republic
Alica.Kelemenova@fpf.slu.cz

**László Lovász**
Department of Computer Science
Eötvös Loránd University
Budapest, Hungary
lovasz@cs.elte.hu

**Gheorghe Păun**
Institute of Mathematics of the
Romanian Academy
Bucharest, Romania
George.Paun@imar.ro

**András Prékopa**
Department of Operations Research
Eötvös Loránd University
Budapest, Hungary
prekopa@cs.elte.hu

**Arto Salomaa**
Department of Mathematics
University of Turku
Turku, Finland
asalomaa@utu.fi

**László Varga**
Department of Software Technology
and Methodology
Eötvös Loránd University
Budapest, Hungary
varga@ludens.elte.hu

**Heiko Vogler**
Department of Computer Science
Dresden University of Technology
Dresden, Germany
vogler@inf.tu-dresden.de

**Gerhard J. Woeginger**
Department of Mathematics and
Computer Science
Eindhoven University of Technology
Eindhoven, The Netherlands
gwoegi@win.tue.nl

# KALMÁR WORKSHOP ON LOGIC AND COMPUTER SCIENCE

*Guest Editors:*

**Ferenc Gécseg**

Department of Computer Algorithms
and Artificial Intelligence
University of Szeged
Szeged, Hungary
gecseg@inf.u-szeged.hu

**György Turán**

Research Group on Artificial Intelligence
of the Hungarian Academy of Sciences
University of Szeged
Szeged, Hungary
turan@inf.u-szeged.hu

Department of Mathematics, Statistics
and Computer Science
University of Illinois at Chicago
Chicago, USA
gyt@uic.edu

# Foreword

This collection of papers is dedicated to László Kalmár (1905 - 1976), a pioneer in mathematical logic, the founder of computer science in Hungary and, among many other accomplishments, the founding editor of *Acta Cybernetica*. It contains the list of his publications, two papers written in his memory and the complete versions of four papers presented at the *Kalmár Workshop on Logic in Computer Science*. The workshop was held in Szeged, October 1-2, 2003. Its complete program can be found at `http://www.inf.u-szeged.hu/kutatas/konferenciak/kalmar2003/` . The workshop was co-located with the *13th International Conference on Inductive Logic Programming (ILP 2003)*. The papers in this collection deal with automata theory, universal algebra, algorithms and computational logic, and thus they are all connected to Kalmár's many-faceted research. We thank the authors for their contributions and Balázs Szörényi for his help.

*Ferenc Gécseg and György Turán*
*Guest Editors*

László Kalmár
(1905–1976)

# In memory of László Kalmár

András Hajnal*

*The following text was presented by Péter Komjáth at the Kalmár Workshop on Logic and Computer Science on October 2, 2003.*

Ladies and Gentlemen,

First, I want to apologize for not being able to attend this meeting.

I was a graduate student of László Kalmár. I arrived to Szeged fifty years ago, almost to the day, having completed my obligatory two months military service after my graduation from the Eötvös University of Budapest. I was supposed to arrive by the morning train and take the tram from the railway station. Professor Kalmár, or Uncle Laci as all the students called him, already wrote me two long letters to the army base describing future plans for the enormous curriculum he planned for me. He was already impatiently waiting for me at the door of the University building. He hardly gave me time to put down my luggage and took me to all the rooms of the Bolyai Institute, introducing me to everybody, while explaining to me Gödel's proof of the consistency of the Generalized Continuum Hypothesis, politely and absent mindedly knocking on all the doors whether we entered the room or left it.

Having described him above as the amiable old professor, let me remind you that he was not yet quite fifty, the same age as your present lecturer Péter Komjáth, my former student, who is supposed to carry on the banner of mathematical logic we handed to him.

To talk seriously, Kalmár was a scientist of enormous authority, a contemporary of Gödel, who was among the very few present at the cradle of Mathematical Logic, who completely understood both the mathematics involved and the significance of it. He had an unparalleled knowledge of contemporary mathematics and he could explain the main points of a subject with deep insight.

Paul Erdős, whose early papers he helped to write, often said that he was a mathematician of von Neumann's caliber. He added that he should have lived in a more fortunate country, where he could have devoted his energies entirely to science.

I am not sure Paul was right. I think and hope that Uncle Laci enjoyed his life, the struggle for his beliefs. His active mind always led him to new discoveries of

*Rényi Institute of Mathematics, Hungarian Academy of Sciences, Reáltanoda utca 13-15, H-1053, Budapest, Hungary

science and that is how he became the founder of Hungarian Computer Science. I think that that is what this conference will mainly be about, so I could stop here, but I want to finish on a personal note.

I am very thankful for all the care and love he and his family gave me while I lived here. I will always cherish the memory of our long mathematical conversations. Later in my life, whenever I did or heard something interesting, I wanted to tell it to him. Sometimes I succeeded, sometimes I did not. I remember, I learned about his death, when returning from abroad I wanted to tell him, that contrary to our earlier intuition there are nontrivial inequalities on cardinal exponentiation.

I am sure this will be the first thing I will tell him when we meet at the place I do not believe in.

Acta Cybernetica 18 (2007) 9–14.

# The activities of László Kalmár in the world of information technology

Árpád Makay*

**Abstract**

Since the end of the 1950s László Kalmár has been interested in the information technology. During a 20 years period he designed several variants of computers interpreting high-level programming languages on architectural levels.

**Keywords:** logical machine, formula-driven computer, high-level language interpreter machine

Towards the end of the 1950s, information technology (IT) became one of the fields in which László Kalmár was highly interested. He was clearly aware of the rapid spread of computers and their excellent applicability for numeral computations. It is nevertheless extremely likely that the links between mathematics and IT were what caught his interest and shaped his views of this field. It is undoubted that he tackled problems from the aspect of a mathematician, always attempting to apply mathematical methods in a world, which at that time was virtually purely technical and technological. It soon became obvious that IT requires and makes wide use of the laws and methods of mathematics: it may suffice merely to mention the inspiring role of automata theory or coding theory. The exactness of mathematics is reflected in the problem solving of IT, the precise understanding of the problems and their detailed analysis, which often demands considerable work. Kalmár's interdisciplinary knowledge played a significant role in his continuous search for new areas of use of IT, defining concrete problems for which he often found solutions and attracted the interest of researchers and developers.

It should be remembered that the freedom of researchers to carry out effective work in Hungary in that period was restricted by a number of factors. The technical resources in the country were rather poor. For understandable reasons, most of the resources were placed in the service of the economy and the running of the state, only a minor part being made available for teaching and research. Kalmár's wide-ranging contacts and (from the 1960s) his nationwide recognition in the world of IT helped him overcome many of the technical obstacles, especially

*University of Szeged, Árpád tér 2, Szeged, Hungary.

in the practical teaching area. Nonetheless, the need to adapt to the external constraints certainly influenced his thoughts and plans. One of the main areas of his interest, the combination programming languages and mathematics, and especially the formal language of logic, was restricted from the outset, the limitations being set by his ambition for the availability of the necessary resources.

Kalmár was unceasingly convinced that the already considerable development of the IT world could benefit still further from the innovative work of Hungarian researchers. In consequence of his international reputation, the leading journals and technical books were at his disposal, often as complimentary copies. The world was generally open to him. At conferences, he was able to meet internationally respected researchers and to set out his ideas and achievements. In this way he acquired up-to-date information on the areas of perspective research and development, which he readily shared with his students and colleagues.

When an effective tool such as the computer becomes available to an individual, his or her imagination suddenly catches wings. And this is what happened to the early IT researchers, engineers and end-users in Hungary, among them László Kalmár. As an example, the machine translation of natural languages seemed attainable from the very beginning. Kalmár closely watched and supported the Hungarian group working on this project. We now know that this goal was reached in part only much later. For that group at that time, the objective appeared unattainable, though their activities furnished important information and knowledge relating to the field of linguistics.

László Kalmár was no stranger to philosophy. Perhaps this was one of the reasons why he became interested in some of the unanswered questions of mathematical logic which (with full mathematical exactness) touched on the limits of reliability of mathematics. Questions often arose in IT (also referred to as cybernetics) such as those concerning the relationship of man and a "thinking" machine, the ability of a machine to reproduce itself, and the controllability of computers. Kalmár developed his own concepts of these issues and often put forward his ideas at appropriate forums. He did this mainly as a mathematician, a stranger to exaggeration and science fiction.

László Kalmár worked in the purely theoretical realm of mathematical logic; it may be stated that he was a real theoretical researcher. In the world of IT, however, he strived towards concrete instruments. The technology was limited, but he designed tools that could be constructed.

The first project that was achieved was a by-product of a departmental seminar. The theme was the technical implementation of mathematical logic (propositional logic). This is an exciting topic if it is considered that the active parts of computers are logical circuits, the tasks of which are logical calculations. The decision was taken to build a "logical machine" that computes the values of logical formulae [1].

The formula applied could contain a maximum of 8 variables, and all the basic operators of calculus could be used. A double contact switch represented the value of a variable or a component formula. One pole had the value TRUE, and the other one the value FALSE. Accordingly, an operator of two variables needed two input and one output switches. Special cables connected the poles. These connections

had to be set according to the logical operators; it may be said that the operators were programmed. The input and output poles of the boxes, already programmed for the basic operators, were appropriately connected to each other so that formulae of desired length became computable. Operator priority and the use of parentheses were enabled. It is obvious that the machine did not tackle the problem by reducing the formula to some kind of normal form. The values of the maximum 8 operands were set by a simple series of switches, their values and that of the formula being indicated by lamps on the display.

The logical machine was a demonstrative tool, but it led to a degree of self-confidence necessary for more difficult challenges to be tackled. Every creator feels the need to explain what his or her creation is good for: the formula built up from the boxes is a circuit that is being tested by the machine. This activity was supported by a relayed "memory", together with the potential for the simultaneous handling of a number of formulae. By 1959 the machine had received a new application (one necessary in most computers): it had become a binary adder.

It was roughly at this time that the training of "programming mathematicians" started at the University of Szeged, and these courses became increasingly more popular. The characteristic features of the courses were very thorough training in mathematics and programming, first in assembly, and later in higher-level languages. While providing several mathematical courses, László Kalmár was the professor of machine programming.

In the 1960s, he experienced that programs written in assembly (or in direct code) were more effective than the codes generated by the compilers, not to mention the time and memory requirements of the compilation process. The effectiveness was a result of the work of the programmer in searching for the memory and time optimum. He also observed that the programming work, i.e. the human energy invested in problem solving, is more effective if a higher-level language is used. This latter is nowadays held to be of greatest importance. At that time, Kalmár could not predict that within 25 years the memory and computing capacity of computers would have become virtually limitless, and that the abilities of compilers would have been enhanced considerably as a consequence of theoretical results. He believed that the solution lay in the approach of machine code to the syntax and semantics of higher-level languages.

During his productive IT activities, Kalmár often returned to this idea of a formula-driven machine. The possibilities available in the various periods are reflected by some of the versions planned throughout the years. In parallel, his goal was a definition of the computer as an algebraic structure, his plans being constructed on this precise theory. As an example, he looked upon a computer operation as (amongst others) a transformation in the memory state. However, because of the large number and complexity of the operations, the characteristics of this transformation could not be written with mathematical exactness. In order for this to be done, the model should have been brought into a much simpler level, e.g. to the level of Turing machine theory, but the practical demands did not allow this. Accordingly, the algebraic model rather played the role of a general approach and was not used directly in the design.

The first plans involved the use of the Ljapunov operational language interpreter machine [2]. The language allowed the use of a limited number of variables, expressions built up with the applications of arithmetical operations, and a few algorithmic tools: conditional clauses and cycles. The syntax was straightforward. Lexical analysis was barely needed. Because of the lack of block structures and program segmentation, there was no need for the most difficult techniques applied in modern interpreters, which at that time were probably impossible to implement with the technical support available then.

The computational unit of the machine was a stack-like structure built up from register quartets. These were designed to handle the arithmetical and logical operations of two operands; they stored the two operands, the operation and (once the values of the operands were available) the result. The result register of the register quartet was connected to both operand registers of the higher-level register quartet through gates. At most one of the gates was open, in order to receive the result of the operation computed at the lower level.

The program was run through the sequential reading of the characters in one pass. At all times there was one active register quartet and one of its registers was active. In one step, the value (if any) of the next variable from the sequence was placed in the active register. In every step, the state of activation of the current register quartet and its register was refreshed, and the states of the gates were set. If the operation could be computed (both operands present in a register quartet), the operation was performed and the result flowed upwards through the open gate.

An analysis of the system reveals that simple cycles can be implemented with the described register hierarchy, since the repeating condition is also an expression. Apart from this, a control unit was needed, with the role of interpretation of the sequential, conditional and cyclic clauses. The memory assignments and the indexed variable handling demanded a special design. Kalmár's plans included all of these features, but the result of prime importance was the technically applicable, special stack architecture. The conditions for the building of the machine could not be met within Hungary, but parts of his machine plans were utilized in the MIR machine of the Ukrainian Academy of Sciences, built in 1966.

By the beginning of the 1960s, it was evident that the stack was an extremely powerful tool in IT. If the traditional infix expressions were converted to postfix form and put into the stack, their interpretation was child's play. The use of a stack simplified the conversion too. If the algorithmic parts of programming languages such as ALGOL-60 could be converted to postfix form, then (by means of a one-pass compilation) the program could be run several times without being compiled in the classical machine language. This was more or less achieved, and extremely efficient interpreters were developed on the basis of this theory. The efficiency was further increased by building the stack into the hardware level of the architecture of the computer, together with the technique of microprogramming.

In the meantime, theoretical results were obtained that gave a definitive direction to the evolution of IT. The design of efficient lexical analysis algorithms was based on the theory of finite automata. The theory of pushdown automata and their special classes defined the limits to be taken into consideration within the syntax

of programming languages for the building of efficient compilers and interpreters. The limitations were loose, and the requirements of the programmers concerning the programming languages could therefore be fully satisfied. The main direction of IT therefore became the use of high-level programming languages (supporting both general needs and special requirements) and the construction of efficient compilers and interpreters for them.

In 1973, László Kalmár was requested by the Hungarian Academy of Sciences to examine the situation regarding computers and higher-level programming languages (including machine languages) and the progress to be expected in these fields. The goal was for Hungarian IT to find its place and to play an appropriate role in the bright future of IT research and development. It was becoming increasingly more obvious that, like all other countries in Eastern Europe, Hungary had failed to recognize the importance of IT in time, and had not allocated sufficient funds for IT research. This invitation was a sign of appreciation of the leader of one of the few research groups which had been producing results and which had come up with constructive ideas despite the inadequate support.

The development of the various generations of computers up to that time, and the methods of constructing computer architectures, were reviewed in a series of monographs [3]. Naturally, the emphasis was on programming languages and their interpretational possibilities, one of Kalmár's main interests. In the final edition, some 15 years after the planning of the first formula-driven machine he put forward a new proposal for a computer that could be programmed through the use of a high-level machine language.

What were the challenges, to which the new plan was intended to respond?

A look at the algorithmic (problem-oriented) languages reveals that the parts building up the syntax have become clear, and new languages can be designed by combining these parts at will. The block structure and the use of modules were necessities. From the aspect of semantics, the notion of the expression had become very clear, mainly as a result of the increased number of data structures. The programs themselves defined complex data types, and the classical sets of values could no longer be used without appropriate care. The definition and implementation were separated, a situation regarded as normal by today's C++ or Java programmer. It was now obvious that the handling of reference types needed extensive redesigning, a feature applied earlier only for indexed variables and memory assignments. It cannot be claimed that the new plan provided adequate answer to all these questions, but it did so for most of them.

The technology too had been evolving. The problems involving the hardware-manipulated stack had been eliminated. The technique of microprogramming was available, a tool that raised the programming level above that of the architecture. The implementation of a redesigned architecture with high-level machine language again seemed feasible.

In the design stage, it became obvious to Kalmár that a single algorithmic language running on a given architecture could no longer satisfy the users. Compilers were clearly needed. The suggested high-level language was designed to be close to more general programming languages, particularly from the aspect of syntax. This

could result in easier compiling, less human work, and lower hardware requirements.

The research group led by Kalmár devised the proposed language. He did not develop a technical plan for the implementation of this language as he had earlier done with the formula-driven machine. He hoped that the various Hungarian work-groups would share their knowledge and that his plans would materialize via such collaboration. In order to verify the language and prove its suitability, he suggested simulation methods. His research group did not lack the necessary knowledge, but the simulator was not built. It is probable that Kalmár's energy was wasted to some extent by his intent on publishing his conceptions in academic and technical circles. The reception was appropriate, and the observations were professional. However, it was at this time that Eastern Europe started planning great developments in the field of IT, and Hungary did not want to be left behind; it therefore adapted to the tendencies determined by "the greats".

What can be said today about formula-driven issues?

Some 30 years ago, IT developed at a very rapid, but quite unexpected tempo. Economizing with hardware resources now belongs to the past; the pace is dictated by the application needs. Many criticize this attitude. Not only has the lowest programming level not risen but it has even become lower, e.g. as a consequence of the RISC technology. The need for the portability of applications over various architectures has nevertheless given rise to a shared-language machine, the Java virtual machine. This has taken place with a different objective and by different means, but its roots are common with those of the formula-driven machine.

# References

[1] L. Kalmár, A new principle of construction of logical machines, in Proceedings of 2-e Congres Unternat. de Cybernetique, Namur (1958) 458-463. See in PDF form: http://www.inf.u-szeged.hu/kalmar2005/tcs/kalmar1958.pdf

[2] L. Kalmár, On a digital computer which can be programmed in a mathematical formula language, in Proceedings of 2nd Hungarian Congress of Mathematics, Budapest (1960) Vol. 5, 3-16. See in PDF form: http://www.inf.u-szeged.hu/kalmar2005/tcs/kalmar1960.pdf

[3] Manuscript series under the title "Belső gépi nyelvek", Szeged (1973) (ed.: László Kalmár). Only available in Hungarian.

*Szeged, 10 December 2004*

# Publications of László Kalmár*

1. On interpolation. *Math. és Phys. Lapok* 33 (1926), 120–149. (In Hungarian.)
2. Zur Theorie der abstrakten Spiele. *Acta Sci. Math.* 4 (1928), 65–85.
3. Über die Abschätzung der Koeffizientensumme Dirichletscher Reihen. *Acta Sci. Math.* 4 (1929), 155–181.
4. Eine Bemerkung zur Entscheidungstheorie. *Acta Sci. Math.* 4 (1929), 248–252.
5. On the problem of "factorisatio numerorum". *Mat. és Fiz. Lapok* 38 (1931), 1–15. (In Hungarian.)
6. Über die mittlere Anzahl der Produktdarstellungen der Zahlen. I. *Acta Sci. Math.* 5 (1931), 95–107.
7. Ein Beitrag zum Entscheidungsproblem. *Acta Sci. Math.* 5 (1932), 222–236.
8. Ein Beweis des Ruffini–Abelschen Satzes. *Acta Sci. Math.* 6 (1932), 59–60.
9. Zum Entscheidungsproblem der mathematischen Logik. *Verhandlungen des Internationalen Mathematiker-Kongresses (Zürich, 1932)*, II. 337–338.
10. Über die Erfüllbarkeit derjenigen Zahlausdrücke, welche in der Normalform zwei benachbarte Allzeichen enthalten. *Math. Ann.* 108 (1933), 466–484.
11. Über einen Löwenheimschen Satz. *Acta. Sci. Math.* 7 (1934), 112–121.
12. Über die Axiomatisierbarkeit des Aussagenkalküls. *Acta Sci. Math.* 7 (1935), 222–243.
13. Zurückführung des Entscheidungsproblems auf den Fall von Formeln mit einer einzigen binären Funktionsvariablen. *Compositio Math.* 4 (1936), 137–144.
14. On the fundamental theorem of arithmetic. *Mat. és Fiz. Lapok* 43 (1936), 27–45. (In Hungarian.)
15. Zur Reduktion des Entscheidungsproblems. *Norsk Mat. Tidsskrift* 19 (1937), 121–130.

*Compiled by András Ádám and Pál Dömösi. Originally appeared in: *Our giants in technology, 6.* Gépipari Tudományos Egyesület, Budapest, 1986. 84-88. (Edited by István Pénzes.) (In Hungarian.)

16. 1938 annual report on the Gyula König Prize. *Mat. és Fiz. Lapok* 45 (1938), 1–17. (In Hungarian.)

17. On the reduction of the decision problem, I: Ackermann prefix, a single binary predicate. *J. Symbolic Logic* 4 (1939), 1–9.

18. On the possibility of definition by recursion. Acta Sci. Math. 9 (1940), 227–232.

19. The objectives, methods and achievements of Hilbert's proof theory. *Mat. és Fiz. Lapok* 48 (1941), 65–119. (In Hungarian.)

20. The development of mathematical exactness from the intuitive to the axiomatic approach. [1] *A másik ember felé, Debrecen (Exodus)* (1942), 39–58. (In Hungarian.)

21. A simple example of an undecidable problem in arithmetic. *Mat. és Fiz. Lapok* 50 (1943), 1–23. (In Hungarian.)

22. A few words about mathematics to those who always hated it, I–II. (In Hungarian.)

    I.: *Pro Christo* 8/4 (1943), 7–9.

    II.: *Pro Christo* 9/3 (1944), 3–5.

23. "Am I hopeless?" *Pro Christo* 8/6 (1943), 7–9. (In Hungarian.)

24. (with János Surányi) On the reduction of the decision problem, II: Gödel prefix, a single binary predicate. *J. Symbolic Logic* 12 (1947), 65–73.

25. On the sum of powers of numbers, I–III. (In Hungarian.)

    I.: *Középisk. Mat. Lapok* 1 (1947–48), 5–10.

    II.: *Középisk. Mat. Lapok* 1 (1947–48), 39–47.

    III.: *Középisk. Mat. Lapok* 1 (1947–48), 169–176.

26. Come, let us prove Chebyshev's theorem!, I–III. (In Hungarian.)

    I.: *Középisk. Mat. Lapok* 1 (1947–48), 89–90.

    II.: *Középisk. Mat. Lapok* 1 (1947–48), 127–128.

    III.: *Középisk. Mat. Lapok* 1 (1947–48), 176–182.

27. Mathematics and dialectical materialism. *Magyar Technika* 3 (1948), 100–102. (In Hungarian.)

28. On unsolvable mathematical problems. *Proceedings of the Tenth International Congress of Philosophy (Amsterdam, 1948)*, 1949, 1. 534–536.

29. Une forme du théorème de Gödel sous des hypothèses minimales. *Comptes Rendus Acad. Sci. Paris* 229 (1949), 963–965.

[1] The printed version of the paper appeared erroneously with "to the axiomatic system" in the title.

30. Quelques formes générales du théorème de Gödel. *Comptes Rendus Acad. Sci. Paris* 229 (1949), 1047–1049.

31. Let us prove Chebyshev's theorem, I–III. (In Hungarian.)

    I.: *Középisk. Mat. Lapok* 2 (1949–50), 7–13.

    II.: *Középisk. Mat. Lapok* 2 (1949–50), 90–91.

    III.: *Középisk. Mat. Lapok* 2 (1949–50), 121–124.

32. (with János Surányi) On the reduction of the decision problem, III: Pepis prefix, a single binary predicate. *J. Symbolic Logic* 15 (1950), 161–173.

33. Eine einfache Konstruktion unentscheidbarer Sätze in formalen Systemen. *Methodos* 2 (1950), 220–226.

34. Another proof of the Gödel–Rosser incompletability theorem. *Acta Sci. Math.* 12 (1950), 38–43.

35. Contributions to the reduction theory of the decision problem, I: Prefix $(x_1)(x_2)(Ex_3)\dots(Ex_{n-1})(x_n)$ a single binary predicate. *Acta Math. Acad. Sci. Hungar.* 1 (1950) 64–73.

36. Über die Cantorsche Theorie der reellen Zahlen. *Publ. Math.* 1 (1950), 150–159.

37. On Cauchy's convergence test. *Acta Math. Acad. Sci. Hungar.* 1 (1950), 109–112.

38. Report on the 2nd World Peace Congress. *Mat. Lapok* 1 (1950), 317–318. (In Hungarian.)

39. Contributions to the reduction theory of the decision problem, [2] III: Prefix $(x_1)(Ex_2)\dots(Ex_{n-2})(x_{n-1})(x_n)$ a single binary predicate. *Acta Math. Acad. Sci. Hungar.* 2 (1951) 19–38.

40. Contributions to the reduction theory of the decision problem, IV: Reduction to the case of a finite set of individuals. *Acta Math. Acad. Sci. Hungar.* 2 (1951), 125–142.

41. (with János Aczél and J. G. Mikusiński) Sur l'équation de translation. *Studia Math.* 12 (1951), 112–116.

42. Another proof of the Markov–Post theorem. *Acta Math. Acad. Sci. Hungar.* 3 (1952), 1–27.

43. Recent results on the foundations of mathematics. *A Magyar Tud. Akad. Mat. Fiz. Oszt. Közl.* 2 (1952), 89–112. (In Hungarian.)

44. Reduction of the decision problem to the question of satisfiability of logical formulas over finite sets. *Az I. Magyar Mat. Kongr. (Budapest, 1950) Közleményei*, 1952, 163–190. (In Hungarian.)

[2] The second part of the series was written by János Surányi.

45. The influence of Bolyai–Lobachevsky geometry on the development of the axiomatic method. *A Magyar Tud. Akad. Mat. Fiz. Oszt. Közl.* 3 (1953), 235–242. (In Hungarian.)

46. The methods of analysis in high school education, I–IV. (In Hungarian.)

    I.: *A mat. tanítása* 1 (1953), 22–32.

    II.: *A mat. tanítása* 1 (1953), 40–50.

    III.: *A mat. tanítása* 1 (1953), 74–80.

    IV.: *A mat. tanítása* 1 (1954), 109–112.

47. L'influence de la géométrie de Bolyai–Lobatchevsky sur le développement de la méthode axiomatique. *Acta. Math. Acad. Sci. Hungar.* 5 (1954), supplementum, 117–126.

48. The solution of a problem of K. Schröter on the definition of the notion of a general recursive function. *A Magyar Tud. Akad. Mat. Fiz. Oszt. Közl.* 5 (1955), 103–127. (In Hungarian.)

49. Über ein Problem, betreffend die Definition des Begriffes der allgemein-rekursiven Funktion. *Zeitschr. f. Math. Logik und Grundlagen d. Math.* 1 (1955), 93–96.

50. On universal algebra—on the borderline between algebra and mathematical logic (abstract of a talk). *Mat. Lapok* 6 (1955), 63–65. (In Hungarian.)

51. A direct proof of the unsolvability of the decision problem by a general recursive algorithm. *A Magyar Tud. Akad. Mat. Fiz. Oszt. Közl.* 6 (1956), 1–25. (In Hungarian.)

52. (with András Hajnal) A remark on Gödel's axiom system for set theory, I–II. (In Hungarian.)

    I.: *Mat. Lapok* 7 (1956), 26–42.

    II.: *Mat. Lapok* 7 (1956), 218–229.

53. (with András Hajnal) An elementary combinatorial theorem with an application to axiomatic set theory. *Publ. Math.* 4 (1956), 431–449.

54. Ein direkter Beweis für die allgemein-rekursive Unlösbarkeit des Entscheidungsproblems des Prädikatenkalküls der ersten Stufe mit Identität. *Zeitschr. f. Math. Logik und Grundlagen d. Math.* 2 (1956), 1–14.

55. Об одной гипотезе, применяемой в исследованиях о так называемых неразрешимых арифметических задачах. *Труды Третьего Всесоюзного Математического Съезда (Москва, 1986)*, 1959, 4, 227–231.

56. On mathematical logic. *Magyar Tudomány* 1 (1956), 369–391. (In Hungarian.)

57. On Church's hypothesis, the foundation of research on so-called unsolvable mathematical problems. *A Magyar Tud. Akad. Mat. Fiz. Oszt. Közl.* 7 (1957), 19–38. (In Hungarian.)

58. A remark on the lecture of Rezső Tarján on "Trends in the development of fast automatic computing machines". *A Magyar Tud. Akad. Mat. Fiz. Oszt. Közl.* 7 (1957), 76–82, 85. (In Hungarian.)

59. Über arithmetische Funktionen von unendlich vielen Variablen, welche an jeder Stelle bloss von einer endlichen Anzahl Von Variablen abhängig sind. *Colloq. Math.* 5 (1957), 1–5.

60. The logic machine of Szeged (abstract of a talk). *Mat. Lapok* 9 (1958), 165. (In Hungarian.)

61. An argument against the plausibility of Church's thesis. *Constructivity in Mathematics (Proc. Coll. Amsterdam, 1957)*, North-Holland, 1959; 72–80.

62. A new principle of construction of logical machines. *2-e Congrès Internat. de Cybernétique (Namur, 1958)*, 458–463.

63. A practical infinitistic computer. *Infinitistic Methods in the Foundations of Mathematics (Proc. Sympos. Warsaw, 1959)*, 1961; 347–362.

64. On a digital computer which can be programmed in a mathematical formula language. *A II. Magyar Matematikai Kongresszus (Budapest, 1960)*, Abstracts, Volume 5., 3–16. (Russian translation: *Кибернетический Сборник, новая серия* 1 (1965), 215–226.)

65. On some graph theoretic problems related to the theory of switching circuits (abstract of a talk). *Mat. Lapok* 11 (1960), 211. (In Hungarian.)

66. Einige philosophische Probleme der Kybernetik. *Naturwissenschaft und Philosophie (Internat. Symp. Leipzig, 1960)*, 381–401.

67. Über einen Rechenautomaten, der eine mathematische Sprache versteht. *Zeitschr. Angew. Math. Mech.* 40 (1960), T, 64–65.

68. Wissenschaftliche Abstraktion und die Anwendung mathematischer Methoden in Biologie und Medizin. *Arzt und Philosophie*, Berlin, 1960; 132–133, 150, 164.

69. Questions of content of the task of university department chairs. *Felsőoktatási Szemle* 10 (1961), 573–580. (In Hungarian.)

70. A contribution to the translation of arithmetical operators (assignment statements) into the machine language of the computer M–3. *Shuxue Jinzhan* 6 (1963), 321–338. (The paper was written in English, but appeared in Chinese.)

71. Algorithmische Sprachen und Programmierung von Rechenautomaten. *Mathematische und Physikalisch-Technische Probleme der Kybernetik (Vorträge der Konferenz, Berlin, 1962)*, 1963; 147–176.

72. Über eine Variante des Neumannschen selbstreproduzierenden Automaten. *Mathematische und Physikalisch-Technische Probleme der Kybernetik (Vorträge der Konferenz, Berlin, 1962)*, 1963; 522–528.

73. Problems of qualitative information theory. *A Magyar Tud. Akad. Mat. Fiz. Oszt. Közl.* 12 (1962), 293–301. (In Hungarian.)

74. Über eine erkenntnistheoretische Wurzel des "Anti-Kybernetismus". *Kybernetik in Wissenschaft*, Berlin, 1962; 53–56.

75. "Conjecture and prove!"? *Magyar Tudomány* 8 (1963), 816–823. (In Hungarian.)

76. Mathematical and linguistic structures. *Általános nyelvészeti tanulmányok, II.* Budapest, 1964; 11–74, 166–172, 295–304. (In Hungarian.)

77. On the problem of the foundation of our knowledge. *The Foundation of Statements and Decisions*, Warsaw, 1965; 13–19.

78. Les calculatrices automatiques comme structures algébriques. *Prévisions, Calcul et Réalités*, Paris, 1965 9–22.

79. О вложении теории автоматических цифровых вычислительных машин в алгебраическую теорию автоматов Мура, Мили и Грушкова. *Теория Конечних и Вероятностных Автоматов*, Москва 1965; 93–99. (A short abstract of this work: On the algebraic theory of automatic digital computers. *Colloquium on the Foundations of Mathematics, Mathematical Machines and their Applications (Tihany, 11-15 September 1962)*, Akadémiai Kiadó, Budapest, 1965; 129.)

80. Un modèle algébrique de calculatrice. *Électronique Troisième Congrès de Calcul et de Traitement de l'Information (Toulouse, 1963)*, Paris, 1965; 381–387.

81. Foundations of mathematics—whither now? *Problems in the Philosophy of Mathematics, (Proc. Coll. London, 1965)*, North-Holland, Amsterdam, 1967; 187–207.

82. Results of mathematical linguistics in the teaching of languages. *Modern Nyelvoktatás* 5 (1967), 25–29. (In Hungarian.)

83. Meaning, synonymy and translation. *Comput. Linguist.* 6 (1967), 27–39.

84. Le langage comme structure algébrique. *Cahiers Linguist. Théor. Appl.* 4 (1967), 73–82.

85. (with Ferenc Obál, István Madarász, Dániel Muszka and György Such) Cybernetical model of the regulation of the homeostasis of the organism. *Abstracts of First Joint Congr. of Hung. Societies of Biochem. Biophys. and Physiol. (Pécs, 1967)*, Budapest, 1967, 42.

86. R. Péter's work in the theory of recursive functions. *Les fonctions recursives et leurs applications (Coll. Internat. Tihany, 1967)*, 1969; 1–11.

87. Pattern recognition and conditional reflexes (general problems). *5-e Congrès Internat. de Cybernétique (Namur, 1967)*, 1968; 136–140.

88. Значение, синонимия и перевод. *Разработка Машинных (Автоматичевских) Систем Перевода с Одного Языка на Другой и их Применение*, Budapest, 1967; 374–390.

89. On the problem of full utilization of the technical possibilities of computers in devising appropriate approximation methods for the solution of numerical problems. *Bull. Math. Soc. Sci. Math. R. S. Roumanie* 12/2 (1968), 75–79.

90. Future tasks concerning programming languages. *Információ és Elektronika* 4 (1968), 251–254. (In Hungarian.)

91. Application of digital computers and special purpose machines in medical diagnostics. *Orvos és Technika* 7 (1969), 14–18. (In Hungarian.)

92. Introduction to cybernetics. *A TIT Fiz. Kém. Mat. Szakosztályának Tájékoztatója* 15 (1969), 55–71. (In Hungarian.)

93. An intuitive representation of context-free languages. *Internat. Conf. Comput. Linguist.* Stockholm, 1969. Preprint 66. (10 pages).

94. On cybernetics. *Fiz. Szemle* 20 (1970), 129–134. (In Hungarian.)

95. Ist ALGOL wirklich eine algoritmische Sprache? *Automatentheorie und Formale Sprachen (Tagung, Oberwolfach, 1969)*, 1970; 305–315.

96. "Fahnendiagramme"—ein anschauliches Hilfsmittel zur Angabe von Programmiersprachen. *Formale Sprachen und Programmiersprachen. (Tagung, Oberwolfach, 1971).*

97. An algebraic model of systems of digital computers. *Internat. Symp. and Summer School on Math. Foundations of Computer Sci.* Warsaw, Jablonna, 1972; 1–12.

98. Problems of the university education of computer scientists. *Felsőoktatási Szemle* 21 (1972), 548–552. (In Hungarian.)

99. Conversations on mathematics. (Interviewer: Özséb Horányi.) *Természet Világa* 103 (1972), 351–356. (In Hungarian.)

100. On a measure of divergence of a context-free language from finite state languages. *Recueil Linguist. de Bratislava IV., Proc. Symp. Algebraic Linguist. (Smolenice, 1970)*, 1973; 93–106.

101. The development of electronic digital computers and the directions of their future development. (Study edited by Kalmár for Section I of the Natural Sciences of the Hungarian Academy of Sciences in 1972. Contributors: Péter Hunya, Ádám Kertész, Pál Quittner, Attila Sára, Sándor Székely.) 1–70. (In Hungarian.)

102. Internal machine languages, including high-level languages. [3] (Study edited by Kalmár for Section I of the Natural Sciences of the Hungarian Academy of Sciences in 1973. Contributors: Éva Gyurkovics, Péter Hunya, Tamás Komor, Árpád Makay, Dániel Muszka, György Révész, Attila Sára, Endre Simon, Sándor Székely, Antal Varga, Tibor Varga.) 1–140. (In Hungarian.)

103. Problems of the education of computer scientists. *A számítástechnikai oktatás a hazai felsőoktatási intézményekben (Conference, Visegrád, 1974)*, Budapest, 1974. 25–30. (In Hungarian.)

[3] Let us mention Chapter 3 of this work and its Appendix. Their titles are: "Proposals for the design of the modern form of Kalmár's formula-directed computer, with special emphasis on its internal langugage", "Syntax of the programming language FCCL–4".

104. Developing a machine independent approach in the education of program designers. *A számítástechnikai oktatás a hazai felsőoktatási intézményekben. (Conference, Visegrád, 1974)*, 142–146. (In Hungarian.)

105. The educator in the age of computers. (Interviewer: László N. Sándor.) *Köznevelés* 30/20 (1974), 3–5. (In Hungarian.)

106. An alternative of the stack-like memory of a computer. (Lecture intended for a conference held in Turku. This posthumous work was edited by Árpád Makay.)

107. *Introduction to mathematical analysis, I–II.* Tankönyvkiadó, Budapest, 1982. (442 + 406 pages). (Edited by Károly Tandori with Piroska Csúri Józsefné Paár, Rozália Dúró Lajosné Lévay, József Németh, Antal Varga.) (In Hungarian.)

Acta Cybernetica 18 (2007) 23–46.

# Groups and Semigroups Defined by some Classes of Mealy Automata

Alexander S. Antonenko* and Eugene L. Berkovich*

**Abstract**

Two classes of finite Mealy automata (automata without branches, slow-moving automata) are considered in this article. We study algebraic properties of transformations defined by automata of these classes. We consider groups and semigroups defined by automata without branches.

**Keywords:** Finite automata; Groups defined by automata; Semigroups defined by automata; Finite automaton transformations.

## Introduction

In this paper we study finite state Mealy automata over two-symbol alphabet and finite state automata transformations defined by them. We shall examine algebraic properties of these transformations, various groups and semigroups of automata transformations and groups defined by noninitial automata of special types.

Groups of automaton transformations have been already investigated in the early sixties of the 20th century (see [1]-[4]). Recent result in the field of semigroups and groups are presented in [6]-[7]. The papers [5] and [8] present reviews of the main results of the theory of automaton transformation groups and semigroups.

Mealy automata turned out to be a convenient tool of defining groups and semigroups. The thing is that small (in number of states and alphabet symbols) Mealy automata generate complex groups.

Those of particular interest are groups with extremal properties, for example, periodic groups of Burnside type, groups of intermediate growth, etc. Mealy automata are used to construct examples of such groups. With their help, Burnside's problem was solved, as well as the problem of intermediate growth groups existence, posed by Milnor in 1968 (the solution of the latter belongs to Grigorchuk).

In the work [10] semigroups and the growth functions of two state automata over two-symbol alphabets are investigated. The question on what groups and semigroups are defined by three state automata over two-symbol alphabets remains unsolved. Therefore, we consider two special classes of automata.

*Odessa I. I. Mechnikov National University. E-mail: {aantonenko,eberk}@mail.ru

The first part of this study sets out the basic definitions and results of Mealy automata theory and gives the definitions of groups and semigroups defined by automata.

The second part is dedicated to Mealy automata over two-symbol alphabets, and a classification of states of such automata is suggested. Two special types of automata are defined on this basis: automata without branches and slow-moving automata.

We obtain results for automata without branches which characterize the groups defined by them for any number of states. Also we study semigroups defined by automata without branches.

The class of slow-moving automata is very wide, and this is why we have limited our investigation to its subclass, namely slow-moving automata of finite type. We have studied the algebraic properties of transformations defined by slow-mowing finite state automata. We have also found family of slow-moving transformations of finite type such that any other one is a composition of members of this family.

# 1 Preliminaries

**Definition 1** ([11, 12]). *A finite Mealy automaton is an ordered quintuple* $A = (X, Y, Q, \pi, \lambda)$, *where* $X$ *is the input alphabet,* $Y$ *is the output alphabet,* $Q$ *is the finite nonempty set of states,* $\pi : X \times Q \to Q$ *is the transition function and* $\lambda : X \times Q \to Y$ *is the output function.* $X$ *and* $Y$ *are finite nonempty sets.*

We will consider only finite automata whose input and output alphabets coincide ($X = Y$). We denote such automata by the quadruples $A = (X, Q, \pi, \lambda)$. Mainly we will consider automata over the two-symbol alphabet $X = \{0, 1\}$.

Let $T_X = \{f | f : X \to X\}$ be the semigroup of all transformations of the set $X$ (the full transformation semigroup), $S_X = \{f | f : X \to X, f \text{ is bijective}\}$ the group of all bijective transformations of the set $X$ (the full symmetric group), $X^*$ the set of all finite words over $X$ and $X^\omega$ the set of all infinite words ($\omega$-words) over $X$.

It is convenient to describe finite automata by the Moore diagrams. We will use the following modification of it. The Moore diagram of an automaton $A$ is an edge-labelled and vertex-labelled directed multigraph $D_A$ with the set of vertices $Q$. Vertices $q_i$ and $q_j$ of the graph $D_A$ are connected by the oriented edge in direction from $q_i$ to $q_j$ marked by the label $x$, if $\pi(x, q_i) = q_j$. Here $x \in X$, $q_i, q_j \in Q$. Every vertex $q$ is labelled by the transformation $\lambda_q \in T_X$ of the alphabet $X$ that corresponds to the output function at the state $q$, i.e. $\lambda_q(x) = \lambda(x, q)$, where $x \in X$, $q \in Q$.

The functions $\pi$ and $\lambda$ can be extended naturally to mappings of the set $X^* \times Q$ into the sets $Q$ and $X^*$ by the following equalities [12]:

$$\begin{aligned} \pi(\Lambda, q) &= q, \quad \pi(wx, q) = \pi(x, \pi(w, q)), \\ \lambda(\Lambda, q) &= \Lambda, \quad \lambda(wx, q) = \lambda(w, q)\lambda(x, \pi(w, q)), \end{aligned}$$

where $\Lambda \in X^*$ is the empty word, $q \in Q$, $w \in X^*$ and $x \in X$. The function $\lambda$

can also be extended in a natural way to a mapping $\lambda : X^\omega \times Q \to X^\omega$ (see for example, [12]).

**Definition 2** ([12]). *The transformation $f_q : X^\omega \to X^\omega$ defined by the equality $f_q(u) = \lambda(u, q)$, where $u \in X^\omega$, is called the automaton transformation defined by the automaton $A = (X, Q, \pi, \lambda)$ at state $q$.*

The Mealy automaton $A = (X, Q, \pi, \lambda)$, where $Q = \{q_0, q_1, \ldots, q_{n-1}\}$, defines the set $F_A = \{f_{q_0}, f_{q_1}, \ldots, f_{q_{n-1}}\}$ of automaton transformations over $X^\omega$.

**Definition 3.** *The Mealy automaton $A$ is called invertible if all transformations from the set $F_A$ are bijections.*

It is easy to show (see for example [5]) that $A$ is invertible if and only if the transformation $\lambda_q$ is a permutation of $X$ for each state $q \in Q$.

**Definition 4** ([12]). *The Mealy automata $A_i = (X, Q_i, \pi_i, \lambda_i)$, $i = 1, 2$, are called isomorphic if there exist two permutations $\xi, \psi \in S_X$ and a one-to-one mapping $\theta : Q_1 \to Q_2$ such that*

$$\theta\pi_1(x, q) = \pi_2(\xi x, \theta q), \qquad \psi\lambda_1(x, q) = \lambda_2(\xi x, \theta q)$$

*for all $x \in X$ and $q \in Q_1$.*

**Definition 5** ([12]). *The Mealy automata $A_i$, $i = 1, 2$, are called equivalent if $F_{A_1} = F_{A_2}$.*

**Proposition 6** ([12]). *Each class of equivalent Mealy automata over the alphabet $X$ contains, up to isomorphism, a unique automaton that is minimal with respect to the number of states (such an automaton is called reduced).*

The minimal automaton can be found using the standard algorithm of minimization.

**Definition 7** ([13]). *For $i = 1, 2$, let $A_i = (X, Q_i, \pi_i, \lambda_i)$ be arbitrary Mealy automata. The automaton $A = (X, Q_1 \times Q_2, \pi, \lambda)$ whose transition and output functions are defined by*

$$\pi\left(x, (q_1, q_2)\right) = \left(\pi_1\left(\lambda_2\left(x, q_2\right), q_1\right), \pi_2\left(x, q_2\right)\right),$$
$$\lambda\left(x, (q_1, q_2)\right) = \lambda_1\left(\lambda_2\left(x, q_2\right), q_1\right),$$

*where $x \in X$ and $(q_1, q_2) \in Q_1 \times Q_2$, is called the* product *of the automata $A_1$ and $A_2$.*

**Proposition 8** ([13]). *For any states $q_1 \in Q_1$, $q_2 \in Q_2$ and arbitrary word $u \in X^*$ the following equality holds:*

$$f_{(q_1,q_2),A}(u) = f_{q_1,A_1}\left(f_{q_2,A_2}(u)\right).$$

**Definition 9.** *The semigroup generated by the set $F_A = \{f_{q_0}, f_{q_1}, \ldots, f_{q_{n-1}}\}$ of transformations defined by a Mealy automaton $A$ in all of its states is called the semigroup defined by the automaton $A$. In the case of an invertible automaton $A$ the group generated by $F_A$ is called the group defined by the automaton $A$.*

## 2 Two special classes of automata

In this section we consider two special classes of automata. We will use the following classification of automata states.

**Definition 10.** *Let $A = (X, Q, \pi, \lambda)$ be a finite automaton. Let us call a state $q \in Q$*

1. *a rest state if for each $x \in X$, $\pi(x, q) = q$ (the automaton will stay in this state)*
2. *an unconditional jump state if there exists a $q' \in Q$, such that $q' \neq q$ and for each $x \in X$, $\pi(x, q) = q'$*
3. *a waiting state if there exists an $x \in X$ such that $\pi(x, q) = q'$, $q' \neq q$ and for each symbol $x' \in X$ with $x' \neq x$, $\pi(x', q) = q$. We will also call this state $x$-waiting state*
4. *a multi-waiting state if there exist $X' \subset X$ and $q' \neq q$ such that $2 \leq |X'| < |X|$ and for each $x' \in X'$, $\pi(x', q) = q'$ and for each $x \notin X'$, $\pi(x, q) = q$*
5. *a conditional jump state or branch state if there exist two distinct symbols $x_1 \neq x_2$ such that $\pi(x_1, q) \neq \pi(x_2, q) \neq q$*

**Definition 11.** *We say that an automaton $A$ is an automaton without branches if all of its states are rest states or unconditional jump ones.*

In other words, the transition function of an automaton without branches depends only on the current state and is independent of input symbols. So for all $q \in Q$ and $x \in X$, we denote $\pi(x, q)$ by $s(q)$.

**Definition 12.** *We call an automaton $A$ slow-moving if all of its states are rest states or waiting ones.*

In other words, for every state $q$, there is at most one symbol $x$ such that $\pi(x, q) \neq q$.

**Definition 13.** *We call a transformation $f : X^\omega \to X^\omega$ slow-moving (without branches) if it can be defined by a slow-moving automaton (without branches).*

**Example 14.** Consider an example of a slow-moving automaton over the two-symbol alphabet $X = \{0, 1\}$ shown in Figure 1. We will consider an infinite input word $w \in X^\omega$ as a 2-adic integer. Let $f$ denote the slow-moving transformation defined by this automaton at the state $q_1$. Then $f$ adds one to any input 2-adic integer. Therefore this automaton is called "adding machine".

Consider the transformation $f^2 = f \circ f$. It is clear that $f^2$ adds two to an input 2-adic integer.

Therefore $f^2$ does not change the first input symbol, and then, not depending on what the first symbol was, acts as transformation $f$ again. Thus, the second symbol is changed, in any case. So the initial state of the automaton defining such transformation can be neither the state of waiting nor the one of rest and the transformation $f^2$ is not slow-moving.

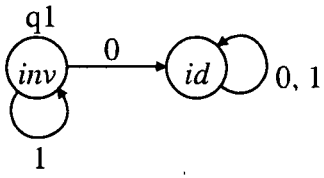


Figure 1: The adding machine

So the product of two slow-moving automata (transformations) is not a slow-moving automaton (transformation) in general.

# 3 Automata without branches

**Definition 15.** *We call the word transformation $f : X^\omega \to X^\omega$ symbol-by-symbol one, if*

$$f(x_1x_2\ldots x_n\ldots) = g_1(x_1)\,g_2(x_2)\ldots g_n(x_n)\ldots$$

*where $g_i : X \to X$.*

**Lemma 16.** *The transformation defined by an automaton without branches is a symbol-by-symbol transformation.*

The proof is clear.

Thus, the transformation $f$ is completely defined by a word $g \in (T_X)^\omega$, $g = g_1g_2\ldots$. Let us denote the corresponding transformation by $F_g$:

$$F_g(x_1x_2\ldots x_n\ldots) = g_1(x_1)\,g_2(x_2)\ldots g_n(x_n)\ldots, \quad g \in X^\omega, \quad g = g_1g_2\ldots g_n\ldots$$

In case $f$ is defined by an invertible automaton over the two-symbol alphabet, each map $g_i$ is either the identity permutation, or transposition. In the first case, we consider $g_i = 0$, in the second one $g_i = 1$.

**Lemma 17.** *Let the transformation $f$ be defined by an automaton without branches with $n$ states. Then $f = F_{uw}$, where $|u| = n$, and $w \in (T_X)^\omega$ is a periodic word. Moreover, the length of the period does not exceed $n$.*

*Proof.* Let $A = (X, Q, \pi, \lambda)$ be an automaton without branches. Then the transformation corresponding to the state $q_k \in Q$ is $F_g$, where $g = g_1g_2\cdots$, $g_{i+1} = \lambda_{s^i(q_k)}$. Recall that $s(q_i) = \pi(x, q_i)$.

Let us consider the sequence $s^i(q_k)$ where $i = 0, 1, 2, \ldots$. Members of this sequence belong to the set $Q = \{q_0, q_1, \ldots, q_{n-1}\}$, which consists of $n$ elements. Hence there are two equal elements $s^p(q_k) = s^{p+l}(q_k)$ among the first $n+1$ ones, where $p < n+1$, $l > 0$, $l \le n$.

Let $r = n - p \ge 0$. Fix an arbitrary $i > 0$. Applying $s^p(q_k) = s^{p+l}(q_k)$, we obtain $s^{r+i}(s^p(q_k)) = s^{r+i}(s^{p+l}(q_k))$. Hence $s^{n+i}(q_k) = s^{n+i+l}(q_k)$.

So the sequence $s^i(q_k)$ is periodic beginning from the member $s^n(q_k)$. It follows that the sequence $g_{i+1} = \lambda_{s^i(q_k)}$ is periodic beginning from the member $g_{n+1}$. The length $l$ of the period does not exceed $n$. □

## 3.1 Groups defined by invertible automata without branches over a two-symbol alphabet

Let us remark that the output function of an invertible automaton over a two-symbol alphabet corresponding to a state $q_i$ is either the identical permutation or the transposition. In the first case we write $\lambda_{q_i} = 0 \in Z_2$. In the second case we write $\lambda_{q_i} = 1 \in Z_2$. Since the transition function $\pi$ of an automaton without branches is independent of any input symbols, we use the notation $s(q_i) = \pi(x, q_i)$. Let us consider $(Z_2)^0$ as the trivial group. The following theorem is applicable:

**Theorem 18.** *Let $U$ be an invertible automaton without branches over a two-symbol alphabet and let $n$ be the number of its states. Then the group defined by it is isomorphic to the group $(Z_2)^r$, where $r = \operatorname{rank} A$,*

$$A = \begin{pmatrix} \lambda_{q_0} & \lambda_{s(q_0)} & \cdots & \lambda_{s^{n-1}(q_0)} \\ \lambda_{q_1} & \lambda_{s(q_1)} & \cdots & \lambda_{s^{n-1}(q_1)} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{q_{n-1}} & \lambda_{s(q_{n-1})} & \cdots & \lambda_{s^{n-1}(q_{n-1})} \end{pmatrix},$$

$$A \in M_n(Z_2), \quad s(q_i) = \pi(x, q_i), \quad x \in X.$$

We first prove some auxiliary lemmas. Let $v^* = vvv\ldots$, where $v \in (Z_2)^n$, $v^* \in (Z_2)^\omega$. We can associate each word $uv$ having the length $n+m$ ($|u| = n$, $|v| = m$) with the map $P_{uv} = F_{uv^*}$.

**Lemma 19.** *The composition of invertible maps $P_{uv}$ and $P_{sw}$ is the map $P_{uv+sw}$, where $u, s \in (Z_2)^n$, $v, w \in (Z_2)^m$, addition is taken modulo 2 like in the group $(Z_2)^{n+m}$.*

*Proof.* The proof is straightforward. □

**Lemma 20.** *Let $U$ be an invertible automaton without branches over a two-symbol alphabet, $n$ the quantity of its states and $m$ the least common multiple of all lengths of the periods of sequences $\left\{s^i(q_k)\right\}_{i=n}^{\infty}$, $k = 0, \ldots, n-1$, $l = n+m$.*

*Then the group defined by $U$ is isomorphic to the group $(Z_2)^r$, where $r = \operatorname{rank} A'$,*

$$A' = \begin{pmatrix} \lambda_{q_0} & \lambda_{s(q_0)} & \cdots & \lambda_{s^{l-1}(q_0)} \\ \lambda_{q_1} & \lambda_{s(q_1)} & \cdots & \lambda_{s^{l-1}(q_1)} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{q_{n-1}} & \lambda_{s(q_{n-1})} & \cdots & \lambda_{s^{l-1}(q_{n-1})} \end{pmatrix},$$

$$A' \in M_{nl}(Z_2), \quad s(q_i) = \pi(x, q_i), \quad x \in X.$$

*Proof.* Let us denote by $G$ the group defined by $U$. Note that all the transformations commute with each other and their orders are equal to 2. So every element of $G$ is a composition of certain transformations $f_i$. Transformation $f_k = P_u$ , where $u$ is the $k$-th row of the matrix $A'$ ($m$ is a period for any sequence $s^i(q_k)$ beginning from $n$-th member). The composition of these transformations $f_i$ is the transformation $P_w$, where $w$ is the sum of the corresponding rows.

Thus, every element of $G$ is a map $P_w$, where $w$ is a linear combination of rows of $A'$ in the linear space $(Z_2)^{n+m}$ over the field $Z_2$ . There are $r$ linearly independent rows among rows of the matrix $A'$. The vector $w$ is uniquely representable in the form of linear combination of $r$ linearly independent rows of the matrix $A'$.

Set one-to-one correspondence between the elements $g \in G$, $g = P_w$, and $r$-vectors of coefficients of linear combination of linear independent rows of the matrix $A'$ representing the vector $w$. Composition operation corresponds to the operation of addition of the coefficient vectors from $(Z_2)^r$.

Thus, $G$ is isomorphic to $(Z_2)^r$. □

*Proof of Theorem 18.* To prove the theorem we need to show that $\operatorname{rank} A = \operatorname{rank} A'$. For this, let $k$ be the minimal number such that the first $k$-1 columns of the matrix $A'$ are linearly independent, but the first $k$ ones are linearly dependent.

Then the $k$-th column is a linear combination of previous columns:

$$A^k = b_1 A^1 + b_2 A^2 + ... + b_{k-1} A^{k-1}, \tag{1}$$

where $A^i$ is a $i$-th column of the matrix $A'$. We can write (1) in a more detailed form:

$$\begin{aligned}
\lambda_{s^{k-1}(q_1)} &= b_1 \lambda_{q_1} + b_2 \lambda_{s(q_1)} + \cdots + b_{k-1} \lambda_{s^{k-2}(q_1)} \\
\lambda_{s^{k-1}(q_2)} &= b_1 \lambda_{q_2} + b_2 \lambda_{s(q_2)} + \cdots + b_{k-1} \lambda_{s^{k-2}(q_2)} \\
&\cdots \\
\lambda_{s^{k-1}(q_n)} &= b_1 \lambda_{q_n} + b_2 \lambda_{s(q_n)} + \cdots + b_{k-1} \lambda_{s^{k-2}(q_n)}
\end{aligned}$$

Let us prove that

$$A^{p+k} = b_1 A^{p+1} + b_2 A^{p+2} + ... + b_{k-1} A^{p+k-1}, \tag{2}$$

for all $p$ from 0 to $l - k$.

Really, fix an arbitrary $i$ between 1 and $n$. Let $s^p(q_i) = q_r$. Then

$$\begin{aligned}
b_1 \lambda_{s^p(q_i)} + b_2 \lambda_{s^{p+1}(q_i)} + ... + b_{k-1} \lambda_{s^{p+k-2}(q_i)} &= b_1 \lambda_{q_r} + b_2 \lambda_{s(q_r)} + ... + b_{k-1} \lambda_{s^{k-2}(q_r)} \\
&= \lambda_{s^{k-1}(q_r)} = \lambda_{s^{p+k-1}(q_i)}
\end{aligned}$$

Thus (2) has been shown. From (2) we can conclude, by induction, that the column $A^{p+k}$ for any $p = 0, \ldots, l - k$ is a linear combination of the columns $A^1$, $A^2$, ..., $A^{k-1}$. Since $k \leq n + 1$, we conclude that $\operatorname{rank} A = \operatorname{rank} A'$. □

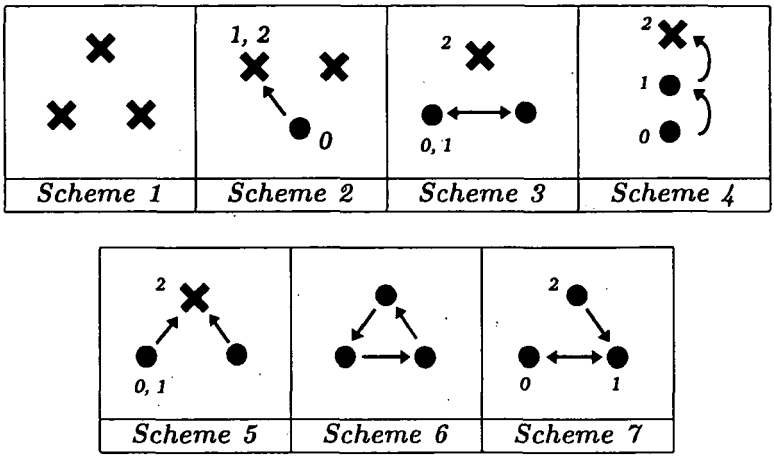


Figure 2: Schemes of transition functions of invertible automata without branches with three states

Theorem 18 allowed us to describe the groups defined by invertible automata without branches with three states.

**Definition 21.** *We call two transition functions* $\pi_1, \pi_2 : X \times Q \to Q$ *equivalent, if there exists a permutation* $\theta \in S_Q$ *such that*

$$\pi_1(x, q) = \theta^{-1}\pi_2(x, \theta(q)) \quad \forall x \in X, q \in Q$$

For automata without branches this equation is $s(q_i) = \theta^{-1}s(\theta(q_i))$.

There are 7 equivalence classes of transition functions of invertible automata without branches with three states. They can be described with the help of schemes (see Figure 2). The cross signs denotes rest states; the dot signs denotes unconditional jump states. The arrows indicate action of transition function. Consider for example automata with transition function corresponding to Scheme 7.

**Scheme 7.** *Let* $t_i = \lambda_{q_i} \in Z_2$.

$$A = \begin{pmatrix} t_0 & t_1 & t_0 \\ t_1 & t_0 & t_1 \\ t_2 & t_1 & t_0 \end{pmatrix}$$

*If* $t_0 = 0$, $t_1 = 0$, $t_2 = 1$, *then the rank equals 1.*
*If* $t_0 = 0$, $t_1 = 1$, $t_2 = 0$, *then the rank equals 2.*
*If* $t_0 = 0$, $t_1 = 1$, $t_2 = 1$, *then the rank equals 3.*
*If* $t_0 = 1$, $t_1 = 0$, $t_2 = 0$, *then the rank equals 2.*
*If* $t_0 = 1$, $t_1 = 0$, $t_2 = 1$, *then the rank equals 2.*
*If* $t_0 = 1$, $t_1 = 1$, $t_2 = 0$, *then the rank equals 2.*
*If* $t_0 = 1$, $t_1 = 1$, $t_2 = 1$, *then the rank equals 1.*

## 3.2 Semigroups defined by automata without branches

Let $v^* = vvv\ldots$, where $v \in (T_X)^n, v^* \in (T_X)^\omega$. We can associate each word $uv$ having the length $n+m$ ($|u| = n$, $|v| = m$) with the map $P_{uv} = F_{uv^*}$.

**Lemma 22.** *The composition of the invertible maps $P_{uv}$ and $P_{sw}$ is the map $P_{uvosw}$, where $u, s \in (T_X)^n, v, w \in (T_X)^m$, and by $\circ$ we denote element by element composition of vectors.*

*Proof.* The proof is straightforward. □

For semigroups defined by automata we can formulate a theorem being a rough analogue to Lemma 20.

**Theorem 23.** *Let $U$ be an automaton without branches and let $n$ be the number of its states. Let $m$ be the least common multiple of all lengths of periods of sequences $\left\{s^i(q_k)\right\}_{i=n}^{\infty}, k = 0, \ldots, n-1$, and let $l = n+m$.*

*Then each transformation defined by $U$ is representable in the form $P_w$, where $w = \left(\lambda_q, \lambda_{s(q)}, \ldots, \lambda_{s^{l-1}(q)}\right) \in (T_X)^l$. Therefore, the semigroup defined by $U$ is isomorphic to the semigroup*

$$sg\left(\left(\lambda_{q_0}, \lambda_{s(q_0)}, \ldots, \lambda_{s^{l-1}(q_0)}\right), \ldots, \left(\lambda_{q_n}, \lambda_{s(q_n)}, \ldots, \lambda_{s^{l-1}(q_n)}\right)\right)$$

*where $sg(g_0, \ldots, g_n)$ is the semigroup generated by $g_0, \ldots, g_n$.*

*Proof.* The semigroup defined by $U$ is generated by the transformations $f_i$, which, by Lemma 17, are representable in the form $F_{uw}$ where $|u| = n$, $w \in X^\omega$ is a periodic word, $uw = \left(\lambda_{q_i}, \lambda_{s(q_i)}, \ldots, \lambda_{s^{l-1}(q_i)}, \ldots\right)$. By the definition of $m$, $f_i$ are representable in the form $P_{uv}$, where $|u| = n$, $|v| = m$. Finally, the isomorphism follows from Lemma 22. □

## 3.3 Semigroups defined by automata without branches over two-symbol alphabets

Automaton transformations over the two-symbol alphabet $X = \{0, 1\}$ are uniquely determined by vectors $u$ of length $l$ the components of which belong to

$$T_X = T_2 = \left\{\alpha = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \beta = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}, id = \varepsilon = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}, inv = \sigma = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\right\}$$

By Lemma 22, the composition of transformations corresponds to the element-by-element composition of vectors. So we reduce study of semigroups defined by automata without branches to study of semigroups of vectors the elements of which belong to $T_2$.

Let $f, g$ be transformations defined by an arbitrary automaton without branches over two-symbol alphabet. The relationships $fff = f$, $fgff = fg$ are true.

We established by numerical experiments that the semigroups of automaton transformations defined by automata without branches with 3 states over the two-symbol alphabet have the following 19 orders (numbers of elements): 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 18, 20, 22, 25, 31. Note that the groups defined by such invertible automata have only one of the following orders: 1, 2, 4, 8.

# 4 Slow-moving automata

The class of slow-moving automata is very wide and it is a rather complicated thing to investigate algebraic properties of transformations defined by slow-moving automata in a general form. That is why we shall consider one more class of automata, namely *automata of finite type* and investigate the transformations defined by slow-moving automata of that class.

## 4.1 Automata of finite type

**Definition 24.** *We call a finite automaton $A$ a finite type automaton if the sequence of automaton states for any infinite input word and for any initial state will stabilize.*

**Definition 25.** *A transformation of infinite words $f : X^\omega \to X^\omega$ we call a finite automaton transformation of finite type if there is a finite type automaton defining the transformation $f$ in some initial state.*

It is rather easy to determine whether the given automaton is a finite type one by its Moore diagram.

**Proposition 26.** *A finite automaton is an automaton of finite type if and only if its Moore diagram is an oriented graph containing no oriented cycles besides the loops.*

*Proof. Necessity.* Let us suppose that the Moore diagram of a finite automaton contains an oriented cycle:

$$q_{i_1}, q_{i_2}, \ldots, q_{i_k}, q_{i_1}$$

Let the automaton start work from the state $q_{i_1}$. Then there is a sequence of input symbols such that the automaton will subsequently be in the states

$$q_{i_1}, q_{i_2}, \ldots, q_{i_k}, q_{i_1}, q_{i_2}, \ldots, q_{i_k}, q_{i_1}, \ldots$$

Therefore, the sequence of states is not stabilized.

*Sufficiency.* Let us take an initial state and a sequence of input symbols. Denote the respective sequence of automaton states by $\{q_{i_k}\}_{k=1}^{\infty}$. If the automaton was in some state $q$ and then went to some other state then it will not be able to return to the state $q$ (since its Moore diagram does not contain oriented cycles besides the loops). Consequently, for each state $q$ there is at most one number $n$ such that $q = q_{i_n} \neq q_{i_{n+1}}$, which means that there are only finitely many numbers $n$ for which $q_{i_n} \neq q_{i_{n+1}}$, that is the sequence $\{q_{i_k}\}_{k=1}^{\infty}$ is stabilized. □

Note that the product of two slow-moving automata (transformations) is not necessarily a slow-moving automaton (transformation), see Example 14. In contrast to the class of slow-moving automata the class of automata of finite type is closed with respect to the product.

**Proposition 27.**

1. *The product of two automata of finite type is an automaton of finite type again.*
2. *The automaton inverse to an invertible automaton of finite type will be of finite type*

*Proof.* Statement 1 follows from the definition of the automata product: if the sequence of the first automaton states is stabilized at the state $q_1$ at the $n$-th step, and that of the second one is stabilized at the state $q_2$ at the $m$-th step, then the sequence of the states of the product is stabilized at the state $(q_1, q_2)$ at the step with number $\max(m, n)$.

Statement 2 Let $A$ be an invertible automaton of finite type. By Proposition 26 its Moore diagram contains no oriented cycles besides the loops. Then the Moore diagram of the inverse automaton of $A$ contains no oriented cycles besides the loops, so it is also an automaton of finite type. □

**Corollary 28.** *The set of all finite automaton transformations of finite type is a subsemigroup of the semigroup of all finite automaton transformations.*

**Corollary 29.** *The set of all invertible finite automaton transformations of finite type is a subgroup of the group of all invertible finite automaton transformations.*

## 4.2 Transformations Defined by Invertible Slow-moving Automata of Finite Type over Two-symbol Alphabets

In this section we shall consider only invertible slow-moving automata of finite type over the two-symbol alphabet $X = \{0, 1\}$. We have studied the algebraic properties of transformations defined by such automata. We have also found a family of slow-moving transformations of finite type such that any other one is a composition of members of this family.

To describe the transformations defined by such automata we shall need special operators acting on the set of all transformations of infinite words $T_{X^\omega} = \{f | f : X^\omega \to X^\omega\}$. Let $p$ be some substitution from the set $S_X = \{id, inv\}$ (here $id$ is an identical substitution, $inv$ is a transposition). For convenience of notation extend the action of $p$ substitution to the sets $X^*$, $X^\omega$ symbol by symbol:

$$p(x_1x_2\ldots x_n) = p(x_1)\,p(x_2)\ldots p(x_n)\,,\ p(x_1x_2\ldots x_n\ldots) = p(x_1)\,p(x_2)\ldots p(x_n)\ldots$$

Let $f \in T_{X^\omega}$. We will denote by $p0]f$ the mapping which acts on an input word as a $p$ substitution up to the first occurrence of zero (including it), and then as an $f$ transformation. We can consider $p0]$ as the operator of the form

$$p0] : T_{X^\omega} \to T_{X^\omega}$$

**Definition 30.** *Let $f \in T_{X^\omega}$. Then $p0]f = g$ is the transformation which acts by the rule*

$$g\,(1^n0w) = p\,(1^n0)\,f\,(w)\,,\ \forall w \in X^\omega, n \geq 0, \quad g\,(1^*) = 1^*$$

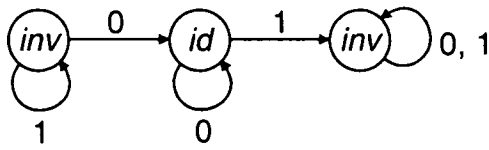


Figure 3: A slow-moving finite state automaton defining the transformation $inv0]id1]inv$.

Here $1^*$ is the infinite word composed of the symbol 1. In other words $g$ acts up to the first zero (including it) by $p$ substitution, and then by $f$ transformation.

The operators

$$p1] : T_{X^\omega} \to T_{X^\omega}$$

are defined similarly.

**Definition 31.** *Let $f \in T_{X^\omega}$. Then $p1]f = g$ is the transformation which acts by the rule*

$$g(0^n 1w) = p(0^n 1) f(w), \forall w \in X^\omega, n \geq 0, \quad g(0^*) = 0^*$$

Let us denote the set of all such operators by $W_G = \{p0], p1] | p \in S_X\}$.

**Example 32.** A slow-moving transformation $s = inv0]id1]inv$ transforms the words from $X^\omega$ as follows. All the symbols up to the first zero (inclusive) are inverted, then until the first one (after the first zero), inclusively, all symbols will remain unchanged, and the rest of the symbols will be inverted again.

This transformation is defined by the automaton shown in Figure 3.

Any transformations defined by *invertible* slow-moving finite state automata can be represented with the help of the above-mentioned operators.

**Proposition 33.** *Let $A$ be a slow-moving invertible finite state automaton. Then any transformation $f$ defined by it can be represented in the form*

$$f = h_1 h_2 \ldots h_k p, \text{ where } h_i \in W_G,\ p \in S_X,\ k \geq 0. \tag{3}$$

*The converse is also true: if the transformation $f$ can be represented in the form (3), then it can be defined by a slow-moving invertible automaton of finite type.*

*Proof.* Let $A$ be an invertible slow-moving automaton of finite type. Remove from its Moore diagram all the loops. Then there will be no more than one arc going from each vertex (since all the states are waiting states or rest states).

In addition the obtained graph will not contain any oriented cycles (since $A$ is an automaton of finite type).

Let us fix some initial state $q_1$ of the automaton. Let us move along the graph beginning from its vertex $q_0$ until we reach the vertex without edges coming from it (sooner or later it will happen since the number of vertices is finite and we cannot be twice in one and the same vertex). While doing it we shall visit vertices

corresponding to the waiting states $q_1, q_2, \ldots, q_k$ and to the rest state $q_{k+1}$, where $k \geq 0$. Let $q_i$ be the $x_i$-waiting state and let the corresponding output function be given by the permutation $p_i$, where $1 \leq i \leq k$. Let $p$ be the output function corresponding to the rest state $q_{k+1}$. Then the transformation $f$ defined by the automaton $A$ in its initial state $q_1$ can be represented in the form $f = h_1 h_2 \ldots h_k p$, where $h_i = p_i x_i]$.

Let us prove the converse statement. Let the transformation $f$ be represented in the form $f = h_1 h_2 \ldots h_k p$, where $h_i = p_i x_i]$. Then the automaton with the $x_i$-waiting states $q_i$ $(1 \leq i \leq k)$ and output functions $p_i$ together with the rest state $q_{k+1}$ and the output function $p$ will define the transformation $f$. □

To formulate the properties of the introduced operators we shall need one more denotation for them. Let $p \in S_X$, $x \in X$. Set

$$px] = \begin{pmatrix} p(x) \\ x \end{pmatrix}.$$

Let us agree that $p^0 = id$, and $p^1 = p$, $p \in S_X$.

**Example 34.** A slow-moving transformation

$$s = inv0]id1]id0]inv1]id1]inv \tag{4}$$

may also be represented in the form

$$s = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} inv. \tag{5}$$

From notation (4) it is clear how exactly the transformation acts, and what automaton defines it. However, notation in the form (5) turns out to be more convenient in many cases, for example, when one has to find a composition of two transformations or turn to the inverted transformation.

**Proposition 35.** *The operators from the set $W_G$ have the following properties:*

1. *Bijective transformation under the action of the operator in the form $p0]$ or $p1]$ turn into a bijective one, and a finite automaton transformation into a finite automaton one.*
2. $px_1]px_2] \ldots px_k]p = p$, $\forall p \in S_X, x_i \in X, i = \overline{1,k}$.
3. $\begin{pmatrix} a \\ b \end{pmatrix} f \circ \begin{pmatrix} b \\ c \end{pmatrix} g = \begin{pmatrix} a \\ c \end{pmatrix} (f \circ g)$, $\forall f, g \in T_{X^\omega}, a, b, c \in X$.
4. $\begin{pmatrix} a \\ a \end{pmatrix} (f \circ g) = \begin{pmatrix} a \\ a \end{pmatrix} f \circ \begin{pmatrix} a \\ a \end{pmatrix} g$, $\forall f, g \in T_{X^\omega}, a \in X$.
5. $\left[ \begin{pmatrix} a \\ b \end{pmatrix} f \right]^{-1} = \begin{pmatrix} b \\ a \end{pmatrix} f^{-1}$, $\forall f \in T_{X^\omega}$, *$f$ is bijective.*

6. $inv^x \circ \begin{pmatrix} a \\ b \end{pmatrix} f \circ inv^y = \begin{pmatrix} a+x \\ b+y \end{pmatrix} (inv^x \circ f \circ inv^y)$, $\forall f \in T_{X^\omega}, a, b, x, y \in X$, *addition here and further on is taken modulo 2.*

*Proof. Property 2* follows directly from the definition of the operator $px]$.

Let us prove *Property 3.* Let $\begin{pmatrix} a \\ b \end{pmatrix} = p_1 b]$, that is $a = p_1(b)$, and $\begin{pmatrix} b \\ c \end{pmatrix} = p_2 c]$, that is $b = p_2(c)$. Let us consider the action of the transformation $\begin{pmatrix} a \\ b \end{pmatrix} f \circ \begin{pmatrix} b \\ c \end{pmatrix} g$ on the word $w \in X^\omega$ in the next two cases

$$1)\ w = \bar{c}^n c w_1 \quad \text{and} \quad 2)\ w = \bar{c}^*,$$

Here and further on $\bar{c}$ is the symbol which is not equal to $c$, i. e. $1 - c$, $\bar{c}^*$ is an infinite word consisting only of the symbol $\bar{c}$, $c \in X$, $w_1 \in X^\omega$

$$1)\ \left[\begin{pmatrix} a \\ b \end{pmatrix} f \circ \begin{pmatrix} b \\ c \end{pmatrix} g\right](\bar{c}^n c w_1) = (p_1 b] f \circ p_2 c] g)(\bar{c}^n c w_1) =$$
$$= (p_1 b] f)(p_2(\bar{c}^n c)\, g(w_1)) = (p_1 b] f)(p_2(\bar{c})^n\, p_2(c)\, g(w_1)) = (*)$$

Note that $p_2(c) = b$, therefore $p_2(\bar{c}) = \bar{b}$ (since $p_2$ is injective). Then $(*) = (p_1 b] f)(\bar{b}^n b g(w_1)) = p_1(\bar{b})^n\, p_1(b) f(g(w_1)) = p_1(p_2(\bar{c}))^n\, p_1(p_2(c)) f(g(w_1)) = [(p_1 \circ p_2)\, c](f \circ g)](\bar{c}^n c w_1) = \left[\begin{pmatrix} p_1(p_2(c)) \\ c \end{pmatrix}(f \circ g)\right](\bar{c}^n c w_1) = \left[\begin{pmatrix} a \\ c \end{pmatrix}(f \circ g)\right](\bar{c}^n c w_1)$

$$2)\ \left[\begin{pmatrix} a \\ b \end{pmatrix} f \circ \begin{pmatrix} b \\ c \end{pmatrix} g\right](\bar{c}^*) = (p_1 b] f \circ p_2 c] g)(\bar{c}^*) = (p_1 b] f)(p_2(\bar{c})^*) =$$
$$(p_1 b] f)(\bar{b}^*) = p_1(\bar{b})^* = p_1(p_2(\bar{c}))^* = ((p_1 \circ p_2)\, c](f \circ g))(\bar{c}^*) =$$
$$\left(\begin{pmatrix} p_1(p_2(c)) \\ c \end{pmatrix}(f \circ g)\right)(\bar{c}^*) = \left(\begin{pmatrix} a \\ c \end{pmatrix}(f \circ g)\right)(\bar{c}^*)$$

*Properties 4 and 5* follow directly from *Property 3.*

To prove *Property 6* we shall use the relationships (6), which follow from *Property 1*:

$$inv^x = \begin{pmatrix} a+x \\ a \end{pmatrix} inv^x; \quad inv^y = \begin{pmatrix} b \\ b+y \end{pmatrix} inv^y \tag{6}$$

From (6), applying *Property 3*, we obtain the required relationship.

The first statement of *Property 1* follows from the already proved *Property 5.*

Let us prove that a finite automaton transformation $f$ under the action of the operator $px] \in W_G$ turns into a finite automaton one. Let $f$ be defined by some finite initial automaton $A_q$ (with initial state $q$).

Let us add to the set of states of this automaton a new state $q_0$. At the same time let us extend the transition function at this state by $\pi(\bar{x}, q_0) = q_0$; $\pi(x, q_0) = q$ and the output function by $\lambda(\bar{x}, q_0) = \lambda(x, q_0) = p(x)$. It is evident that $q_0$ will be an $x$-waiting state. Let us choose this state the initial one. Then the obtained initial automaton $A'_{q_0}$ will determine the transformation $f$. □

It follows from Property 2 of Proposition 35 that the representation (3) for slow-moving transformation of finite type is not single-valued but it could be always brought to the form:

$$p_1x_1]p_2x_2]\dots p_kx_k]p, \text{ where } p \neq p_k \quad p, p_i \in S_X, \quad x_i \in X, \quad i = \overline{1,k} \tag{7}$$

Let us call the representation (7) *canonical.*

**Proposition 36.** *Every slow-moving transformation of finite type has exactly one canonical representation.*

*Proof.* Assume that the transformation $f$ have two different canonical representations:

$$f = p_1x_1]p_2x_2]\dots p_kx_k]p = p'_1x'_1]p'_2x'_2]\dots p'_{k'}x'_{k'}]p'$$

Let us suppose that there exists a number $l$ such that $\forall i < l:\ p_i = p'_i,\ x_i = x'_i$, and $p_l \neq p'_l$ or $x_l \neq x'_l$. Otherwise we have $k \neq k'$ (without loss of generality we may assume $k < k'$) and $\forall i = \overline{1,k}:\ p_i = p'_i,\ x_i = x'_i$. This case will be considered later.

Note that the situation $k = k',\ \forall i = \overline{1,k}:\ p_i = p'_i,\ x_i = x'_i$ and $p \neq p'$ is impossible since one of the representations will not be canonical.

If $p_l \neq p'_l$, it is easily seen that

$$\begin{aligned} f(x_1x_2\dots x_{l-1}aw) &= (p_1x_1]p_2x_2]\dots p_kx_k]p)(x_1x_2\dots x_{l-1}aw) = \\ &= p_1(x_1)p_2(x_2)\dots p_{l-1}(x_{l-1})p_l(a)u \\ f(x_1x_2\dots x_{l-1}aw) &= (p'_1x'_1]p'_2x'_2]\dots p'_kx'_k]p')(x_1x_2\dots x_{l-1}aw) = \\ &= p'_1(x_1)p'_2(x_2)\dots p'_{l-1}(x_{l-1})p'_l(a)u' \end{aligned}$$

where $a \in X$, $w, u, u' \in X^\omega$. This is impossible since $p_l(a) \neq p'_l(a)$. If $p_l = p'_l = p_0$, then we shall find a maximal number $m$ such that $p_0 = p_l = p_{l+1} = \dots = p_m, \quad m \leq k$ (if $m < k$, then $p_m \neq p_{m+1}$).

Similarly, $m'$ is a maximal number such that $p'_l = p'_{l+1} = \dots = p'_{m'}$. Let us assume that $m - l \leq m' - l$, the case $m - l \geq m' - l$ can be treated in a similar way. Then it is not difficult to see that

$$\begin{aligned} f(x_1x_2\dots x_{l-1}x_l\dots x_maw) &= (p_1x_1]p_2x_2]\dots p_kx_k]p)(x_1x_2\dots x_{l-1}x_l\dots x_maw) = \\ &= p_1(x_1)p_2(x_2)\dots p_{l-1}(x_{l-1})p_0(x_l\dots x_m)r(a)u, \end{aligned}$$

where $r = p_{m+1}$ if $m < k$, and $r = p$ if $m = k$ ($a \in X$, $w, u, u' \in X^\omega$). On the other hand

$$\begin{aligned} f(x_1x_2\dots x_{l-1}x_l\dots x_maw) &= (p'_1x'_1]p'_2x'_2]\dots p'_{k'}x'_{k'}]p')(x_1x_2\dots x_{l-1}x_l\dots x_maw) = \\ &= p'_1(x_1)p'_2(x_2)\dots p'_{l-1}(x_{l-1})p_0(x_l\dots x_m)p_0(a)u' \end{aligned} \tag{8}$$

which is impossible since $p_0(a) \neq r(a)$.

We need only consider the case when $k < k'$ and $\forall i = \overline{1,k}: p_i = p'_i,\ x_i = x'_i$. There are two subcases:

1. $\exists s > k : p'_s \neq p$ and
2. $(\forall s > k : p'_s = p) \,\&\, (p' \neq p)$

In the first subcase let $s$ be the minimal number such that $p'_s \neq p$. In the word $f(x'_1 x'_2 \dots x'_s a w)$ the symbol with number $s{+}1$ will be $p'_s(a)$ on one side and $p(a)$ on the other side. In the second subcase in the word $f(x'_1 x'_2 \dots x'_k a w)$ the symbol with number $k{+}1$ will be $p'(a)$ on one side and $p(a)$ on the other side. Therefore, in any cases we obtain a contradiction. □

Let us consider a family of slow-moving transformations of finite type:

$$\alpha_0 = inv, \quad \alpha_1 = id0]inv, \quad \alpha_2 = id0]id0]inv, \quad \dots, \quad \alpha_n = id0]^n inv, \quad \dots$$

All the $\alpha_i$ are the involutions, that is $\alpha_i^2 = id$. We will show that all the slow-moving transformations of finite type can be represented in the form of compositions of $\alpha_i$.

**Theorem 37.** *The following equality holds*

$$\begin{pmatrix} a_0 \\ b_0 \end{pmatrix} \begin{pmatrix} a_1 \\ b_1 \end{pmatrix} \dots \begin{pmatrix} a_{n-1} \\ b_{n-1} \end{pmatrix} inv^{a_n} =$$
$$= \alpha_0^{a_0} \alpha_1^{a_0+a_1} \alpha_2^{a_1+a_2} \dots \alpha_{n-1}^{a_{n-2}+a_{n-1}} \alpha_n^{a_{n-1}+a_n+b_{n-1}} \alpha_{n-1}^{b_{n-1}+b_{n-2}} \dots \alpha_2^{b_2+b_1} \alpha_1^{b_1+b_0} \alpha_0^{b_0}$$
$$n \geq 1 \quad (9)$$

*Proof.* The proof will be made by induction on $n$.
*Base of induction*: $n = 1$.

Applying Property 6 of Proposition 35, we obtain

$$\begin{pmatrix} a_0 \\ b_0 \end{pmatrix} inv^{a_1} = inv^{a_0} \circ inv^{a_0} \circ \begin{pmatrix} a_0 \\ b_0 \end{pmatrix} inv^{a_1} \circ inv^{b_0} \circ inv^{b_0} =$$
$$= inv^{a_0} \circ \begin{pmatrix} a_0 + a_0 \\ b_0 + b_0 \end{pmatrix} \left(inv^{a_0} \circ inv^{a_1} \circ inv^{b_0}\right) \circ inv^{b_0} =$$
$$= \alpha_0^{a_0} \circ \begin{pmatrix} 0 \\ 0 \end{pmatrix} \left(inv^{a_0+a_1+b_0}\right) \circ \alpha_0^{b_0} = \alpha_0^{a_0} \circ \alpha_1^{a_0+a_1+b_0} \circ \alpha_0^{b_0}$$

If $a_0{+}a_1{+}b_0 = 0$, then $\begin{pmatrix} 0 \\ 0 \end{pmatrix} \left(inv^{a_0+a_1+b_0}\right) = id$, otherwise $\begin{pmatrix} 0 \\ 0 \end{pmatrix} \left(inv^{a_0+a_1+b_0}\right) = \alpha_1$.

*Transition of induction*: Suppose that the statement of the theorem is valid for $n = k - 1$. Let us prove it for $n = k$:

$$\begin{pmatrix} a_0 \\ b_0 \end{pmatrix} \begin{pmatrix} a_1 \\ b_1 \end{pmatrix} \dots \begin{pmatrix} a_{k-1} \\ b_{k-1} \end{pmatrix} inv^{a_k} =$$
$$= inv^{a_0} \circ inv^{a_0} \circ \begin{pmatrix} a_0 \\ b_0 \end{pmatrix} \begin{pmatrix} a_1 \\ b_1 \end{pmatrix} \dots \begin{pmatrix} a_{k-1} \\ b_{k-1} \end{pmatrix} inv^{a_k} \circ inv^{b_0} \circ inv^{b_0} =$$

Applying Property 6 of Proposition 35, we obtain

$$= inv^{a_0} \circ \binom{a_0 + a_0}{b_0 + b_0} \left[ inv^{a_0} \circ \binom{a_1}{b_1} \cdots \binom{a_{k-1}}{b_{k-1}} inv^{a_k} \circ inv^{b_0} \right] \circ inv^{b_0} =$$

Applying the assumption of induction:

$$= inv^{a_0} \circ \binom{0}{0} \left[ inv^{a_0} \circ \alpha_0^{a_1} \alpha_1^{a_1+a_2} \alpha_2^{a_2+a_3} \ldots \alpha_{k-2}^{a_{k-2}+a_{k-1}} \alpha_{k-1}^{a_{k-1}+a_k+b_{k-1}} \circ \right.$$
$$\left. \circ \alpha_{k-2}^{b_{k-1}+b_{k-2}} \ldots \alpha_2^{b_3+b_2} \alpha_1^{b_2+b_1} \alpha_0^{b_1} \circ inv^{b_0} \right] \circ inv^{b_0} =$$
$$= \alpha_0^{a_0} \circ \binom{0}{0} \left[ \alpha_0^{a_0+a_1} \alpha_1^{a_1+a_2} \alpha_2^{a_2+a_3} \ldots \alpha_{k-2}^{a_{k-2}+a_{k-1}} \alpha_{k-1}^{a_{k-1}+a_k+b_{k-1}} \circ \right.$$
$$\left. \circ \alpha_{k-2}^{b_{k-1}+b_{k-2}} \ldots \alpha_2^{b_3+b_2} \alpha_1^{b_2+b_1} \alpha_0^{b_1+b_0} \right] \circ \alpha_0^{b_0} =$$

Applying Property 4 of Proposition 35 and the relationship $\binom{0}{0} \alpha_i = \alpha_{i+1}$, we obtain

$$= \alpha_0^{a_0} \alpha_1^{a_0+a_1} \alpha_2^{a_1+a_2} \ldots \alpha_{k-1}^{a_{k-2}+a_{k-1}} \alpha_k^{a_{k-1}+a_k+b_{k-1}} \alpha_{k-1}^{b_{k-1}+b_{k-2}} \ldots \alpha_2^{b_2+b_1} \alpha_1^{b_1+b_0} \alpha_0^{b_0}.$$

□

Thus, a slow-moving transformation of finite type can be represented as follows:

$$s = \alpha_{i_1} \alpha_{i_2} \ldots \alpha_{i_k}, \tag{10}$$

where

(10.1) $i_r \neq i_{r+1}$ for all $r = \overline{1,k}$ and

(10.2) there exists an $m$, so that $i_p < i_q$, if $p < q \leq m$, and $i_p > i_q$, if $m \leq p < q$.

On the contrary, if $\{i_r\}_{r=1}^{k}$ is the sequence of nonnegative integers satisfying conditions (10.1) and (10.2), then it follows from Theorem 37 that the transformation $s = \alpha_{i_1} \alpha_{i_2} \ldots \alpha_{i_k}$ is slow-moving of finite type (it is not difficult to select the corresponding $a_i$, $b_i \in Z_2$).

## 4.3 Noninvertible slow-moving automata of finite type

Let us consider the slow-moving automata of finite type over the two-symbol alphabet $X = \{0, 1\}$ being a generalization of the corresponding invertible automata studied in Section 4.2. To describe the transformations defined by such automata, we need to extend the set of the operators considered in Section 4.2.

Let $p$ be some transformation from the set $T_X = T_2 = \{id = \varepsilon, inv = \sigma, \alpha, \beta\}$. Extend the action of the transformation $p$ to the sets $X^*$ and $X^\omega$ symbol by symbol as in Section 4.2.

The operators $p0]$ and $p1]$ are introduced similarly as in Section 4.2:

$$p0] : T_{X^\omega} \to T_{X^\omega}, \quad p0]f = g,$$

where $g$ acts according to the rule

$$g(1^n 0w) = p(1^n 0) f(w), \forall w \in X^\omega, n \geq 0, \quad g(1^*) = p(1^*)$$

and

$$p1] : T_{X^\omega} \to T_{X^\omega}, \quad p1]f = g,$$

where $g$ acts according to the rule

$$g(0^n 1w) = p(0^n 1) f(w), \forall w \in X^\omega, n \geq 0, \quad g(0^*) = p(0^*).$$

Set $W_S = \{px] | p \in T_2, x \in X\} = \{p0], p1] | p \in T_2\}$. It is evident that $W_G \subset W_S$.

**Proposition 38.** *Let $A$ be a slow-moving automaton of finite type. Then any transformation $f$ defined by it can be represented in the form*

$$f = h_1 h_2 \ldots h_k p, \text{ where } h_i \in W_S, p \in T_2, k \geq 0 \tag{11}$$

*The inverse statement is also true: if the transformation $f$ can be represented in the form (11), then it can be defined by a slow-moving automaton of finite type.*

*Proof.* The proof is similar to that of Proposition 33. □

Let us introduce one more notation for the operators from $W_S$:

$$px] = \begin{pmatrix} a \\ p(x) \\ x \end{pmatrix}, a = \begin{cases} 1, & p \in \{\alpha, \beta\} \\ 0, & p \in \{id, inv\} \end{cases}$$

The notation of the form $\begin{pmatrix} 0 \\ a \\ b \end{pmatrix}$ corresponds to the notation $px] = \begin{pmatrix} a \\ b \end{pmatrix} \in W_G$ of Section 4.2. Set $p^0 = id$, and $p^1 = p$, $p \in T_2$. Let $\overline{x} = 1 - x$, $x \in X = \{0, 1\}$.

**Proposition 39.** *The following properties hold for the operators from $W_S$:*

1. *Finite automaton transformations turn into finite automaton transformations under the action of operators of the form $p0]$ or $p1]$.*
2. *$px_1]px_2] \ldots px_k]p = p$, for all $p \in T_2, x_i \in X, i = \overline{1,k}$.*
3. *for all $g \in T_{X^\omega}$, $a, b \in X$, $x \in X^\omega$, $n \geq 0$*

$$\left(\begin{pmatrix} 0 \\ a \\ b \end{pmatrix} g\right)\left(\overline{b}^n bx\right) = \overline{a}^n a g(x), \quad \left(\begin{pmatrix} 0 \\ a \\ b \end{pmatrix} g\right)\left(\overline{b}^*\right) = \overline{a}^*,$$

4. *for all* $g \in T_{X^\omega}$, $a, b \in X$, $x \in X^\omega$, $n \geq 0$

$$\left(\begin{pmatrix}1\\a\\b\end{pmatrix} g\right)\left(\overline{b}^n bx\right) = a^{n+1} g(x), \quad \left(\begin{pmatrix}1\\a\\b\end{pmatrix} g\right)\left(\overline{b}^*\right) = a^*$$

5. $\begin{pmatrix}d\\a\\b\end{pmatrix} f \circ \begin{pmatrix}0\\b\\c\end{pmatrix} g = \begin{pmatrix}d\\a\\c\end{pmatrix}(f \circ g), \quad \forall f, g \in T_{X^\omega}, a, b, c \in X.$

6. $inv^x \circ \begin{pmatrix}d\\a\\b\end{pmatrix} f \circ inv^y = \begin{pmatrix}d\\a+x\\b+y\end{pmatrix}(inv^x \circ f \circ inv^y), \quad \forall f \in T_{X^\omega}, a, b, x, y \in X,$
*the addition here and further on is taken modulo 2.*

*Proof.*

1. The proof is similar to that of Property 1 for the operators from $W_G$.
2. The proof follows from the definition of $px]$.
3. Let $p \in \{id, inv\}$, $p(b) = a$, $p\left(\overline{b}\right) = \overline{a}$. Then $pb] = \begin{pmatrix}0\\a\\b\end{pmatrix}$, hence from the definition of $pb]$ the property follows.
4. Let $p \in \{\alpha, \beta\}$, $p(b) = p\left(\overline{b}\right) = a$. Then $pb] = \begin{pmatrix}1\\a\\b\end{pmatrix}$, hence from the definition of $pb]$ the property follows.
5. Let us consider the action of the left and right sides of equality on words of the form $\overline{c}^n cx$, where $n \geq 0$, $x \in X^\omega$ and $c^* = cc\ldots$
   Using properties 3 and 4 we obtain:

$$\left[\begin{pmatrix}d\\a\\b\end{pmatrix} f \circ \begin{pmatrix}0\\b\\c\end{pmatrix} g\right](\overline{c}^n cx) = \begin{pmatrix}d\\a\\b\end{pmatrix} f\left[\begin{pmatrix}0\\b\\c\end{pmatrix} g(\overline{c}^n cx)\right] = \begin{pmatrix}d\\a\\b\end{pmatrix} f\left(\overline{b}^n bg(x)\right) =$$

$$= \begin{cases}\overline{a}^n a f(g(x)), & d = 0\\ a^{n+1} f(g(x)), & d = 1\end{cases} = \begin{cases}\overline{a}^n a (f \circ g)(x), & d = 0\\ a^{n+1}(f \circ g)(x), & d = 1\end{cases} = \begin{pmatrix}d\\a\\c\end{pmatrix}(f \circ g)(\overline{c}^n cx)$$

$$\left[\begin{pmatrix}d\\a\\b\end{pmatrix} f \circ \begin{pmatrix}0\\b\\c\end{pmatrix} g\right](\overline{c}^*) = \begin{pmatrix}d\\a\\b\end{pmatrix} f\left[\begin{pmatrix}0\\b\\c\end{pmatrix} g(\overline{c}^*)\right] = \begin{pmatrix}d\\a\\b\end{pmatrix} f\left(\overline{b}^*\right) =$$

$$= \begin{cases}\overline{a}^*, & d = 0\\ a^*, & d = 1\end{cases} = \begin{pmatrix}d\\a\\c\end{pmatrix}(f \circ g)(\overline{c}^*)$$

6. By Property 2, $\begin{pmatrix} 0 \\ a+x \\ a \end{pmatrix} inv^x = inv^x a]inv^x = inv^x$; $\begin{pmatrix} 0 \\ b \\ b+y \end{pmatrix} inv^y = inv^y b]iny^y = inv^y$. We have $\begin{pmatrix} d \\ a \\ b \end{pmatrix} f \circ inv^y = \begin{pmatrix} d \\ a \\ b \end{pmatrix} f \circ \begin{pmatrix} 0 \\ b \\ b+y \end{pmatrix} inv^y = \begin{pmatrix} d \\ a \\ b+y \end{pmatrix} (f \circ inv^y)$. If $x = 0$, then Property 6 turns into the last equality. Otherwise, if $d = 0$ then Property 5 gives

$$inv^x \circ \begin{pmatrix} 0 \\ a \\ b \end{pmatrix} f \circ inv^y = \begin{pmatrix} 0 \\ a+x \\ a \end{pmatrix} inv^x \circ \begin{pmatrix} 0 \\ a \\ b+y \end{pmatrix} (f \circ inv^y) =$$

$$= \begin{pmatrix} 0 \\ a+x \\ b+y \end{pmatrix} (inv^x \circ f \circ inv^y)$$

If $d = 1, x = 1$, then $inv \circ \begin{pmatrix} 1 \\ a \\ b \end{pmatrix} f \circ inv^y = inv \circ \begin{pmatrix} 1 \\ a \\ b+y \end{pmatrix} (f \circ inv^y)$. By the definitions of the operators $\begin{pmatrix} 1 \\ a \\ b+y \end{pmatrix}$ and $\begin{pmatrix} 1 \\ a+1 \\ b+y \end{pmatrix}$ we obtain $inv \circ \begin{pmatrix} 1 \\ a \\ b+y \end{pmatrix} (f \circ inv^y) = \begin{pmatrix} 1 \\ a+1 \\ b+y \end{pmatrix} (inv \circ f \circ inv^y)$, which proves Property 6.

□

Similarly as in Section 4.2, we can introduce the notion of the canonical representation of an arbitrary (not necessarily invertible) slow-moving finite automaton transformation of finite type which is unique.

Let

$$\alpha_0 = inv, \quad \alpha_1 = id0]inv, \quad \alpha_2 = id0]id0]inv, \quad \ldots, \quad \alpha_n = id0]^n inv, \quad \ldots$$

$$\beta_1 = \alpha 0]id, \beta_2 = id0]\alpha 0]id, \beta_3 = id0]id0]\alpha 0]id, \ldots, \beta_n = id0]^{n-1}\alpha 0]id, \ldots$$

$$\gamma_1 = \alpha 0]inv, \gamma_2 = id0]\alpha 0]inv, \gamma_3 = id0]id0]\alpha 0]inv, \ldots, \gamma_n = id0]^{n-1}\alpha 0]inv, \ldots$$

$$\delta_0 = \alpha, \quad \delta_1 = id0]\alpha, \quad \delta_2 = id0]id0]\alpha, \quad \ldots, \quad \delta_n = id0]^n\alpha, \quad \ldots$$

$$\lambda_{1,i} = \alpha_i, \lambda_{2,i} = \beta_{i+1}, \lambda_{3,i} = \gamma_{i+1}, \lambda_{4,i} = \delta_i, i \geq 0$$

It is evident that we have $\lambda_{j,i+1} = id0]\lambda_{j,i} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \lambda_{j,i}, i \geq 0, 1 \leq j \leq 4$.

All $\alpha_i$ are involutions, $\alpha_i^2 = id$, all $\beta_i, \delta_i$ are idempotents, that is $\beta_i^2 = \beta_i, \delta_i^2 = \delta_i$. It is clear that $\alpha_0^2 = id$, $\delta_0^2 = \delta_0$. Let us prove the idempotency of $\beta_1$. We have $\beta_1(1^n 0x) = 0^{n+1}x$, $\beta_1^2(1^n 0x) = \beta_1(0^{n+1}x) = \beta_1(00^n x) = 00^n x = \beta_1(1^n 0x)$, $\beta_1(1^*) = 0^* = 00^*, \beta_1^2(1^*) = \beta_1(00^*) = 00^* = \beta_1(1^*)$. Then, $\alpha_i^2 = id0]^i(\alpha_0^2) = id0]^i id = id$, $\beta_i^2 = id0]^{i-1}(\beta_1^2) = id0]^{i-1}\beta_1 = \beta_i$, $\delta_i^2 = id0]^i(\delta_0^2) = id0]^i\delta_0 = \delta_i$. It is evident that $\gamma_i$ are not idempotents.

Theorem 37 can be generalized to the following: all the slow-moving transformations of finite type can be represented in the form of compositions of $\alpha_i, \beta_i, \gamma_i, \delta_i$ (or which is the same $\lambda_{j,i}, i \geq 0, 1 \leq j \leq 4$).

**Theorem 40.** *Any slow-moving transformation of finite type $f = h_1 h_2 \dots h_k p$, where $h_i \in W_S$, $p \in T_2$, $k \geq 0$, can be represented in the form*

$$f = f_1 \circ f_2 \circ \dots \circ f_r,\ r > 0,\ f_j \in \{\lambda_{s,i} | i \geq 0, 1 \leq s \leq 4\},\ j = \overline{1,r} \qquad (12)$$

*More exactly, if* $h_i = \begin{pmatrix} c_{i-1} \\ b_{i-1} \\ a_{i-1} \end{pmatrix}$, $a_{i-1}, b_{i-1}, c_{i-1} \in X$, $i = \overline{1,k}$ *then*

$$f = L_0(c_0, a_0, b_0) \circ L_1(c_1, a_1, b_1) \circ \dots \circ L_{k-1}(c_{k-1}, a_{k-1}, b_{k-1}) \circ C_k(p) \circ \\ \circ R_{k-1}(b_{k-1}) \circ \dots \circ R_1(b_1) \circ R_0(b_0) \qquad (13)$$

*where*

$$L_i(c,a,b) = \begin{cases} \alpha_i^a \circ \alpha_{i+1}^a, & \text{if } c = 0 \\ \alpha_i^a \circ \beta_{i+1} \circ \alpha_{i+1}^b, & \text{if } c = 1, a+b = 0 \\ \alpha_i^a \circ \gamma_{i+1} \circ \alpha_{i+1}^b, & \text{if } c = 1, a+b = 1 \end{cases}$$

$$R_i(b) = \alpha_{i+1}^b \circ \alpha_i^b$$

$$C_i(p) = \begin{cases} id, & \text{if } p = id \\ \alpha_i, & \text{if } p = inv \\ \delta_i, & \text{if } p = \alpha \\ \alpha_i \circ \delta_i, & \text{if } p = \beta \end{cases}$$

*The form of the function (13) turns into the form of the function (12) by throwing id from the composition (13), except for the case $f = id$.*

*Proof.* We will prove the theorem by induction on the number k.

The base of induction. Let $k = 0$. Then $f = p \in \{id, inv, \alpha, \beta\}$, and

$$id = C_0(id) = \alpha_0 \circ \alpha_0, \quad inv = \alpha_0 = C_0(inv), \quad \alpha = \delta_0 = C_0(\alpha), \\ \beta = inv \circ \alpha = \alpha_0 \circ \delta_0 = C_0(\beta)$$

that is $f$ can be represented in forms (12) and (13).

Suppose that the statement of the theorem holds for $k = l$ and prove it for $k = l + 1$. Let $g = h_2 h_3 \dots h_k p$. Then $f = h_1 g$ and by the induction hypothesis g

can be represented as $g = L_0(c_1, a_1, b_1) \circ L_1(c_2, a_2, b_2) \circ \cdots \circ L_{k-2}(c_{k-1}, a_{k-1}, b_{k-1}) \circ C_{k-1}(p) \circ R_{k-2}(b_{k-1}) \circ \cdots \circ R_0(b_1)$. Note that $id0]L_i(c, a, b) = L_{i+1}(c, a, b)$, $id0]R_i(b) = R_{i+1}(b)$, $id0]C_i(p) = C_{i+1}(p)$, therefore

$$id0]g = L_1(c_1, a_1, b_1) \circ L_2(c_2, a_2, b_2) \circ \cdots \circ L_{k-1}(c_{k-1}, a_{k-1}, b_{k-1}) \circ C_k(p) \circ \\ \circ R_{k-1}(b_{k-1}) \circ \cdots \circ R_1(b_1)$$

There are two cases to consider:

Let $h_1 = \begin{pmatrix} 0 \\ a \\ b \end{pmatrix}$. Then

$$\begin{pmatrix} 0 \\ a \\ b \end{pmatrix} g = inv^a \circ \left( inv^a \circ \begin{pmatrix} 0 \\ a \\ b \end{pmatrix} g \circ inv^b \right) \circ inv^b =$$

$$= inv^a \circ \left( \begin{pmatrix} 0 \\ a+a \\ b+b \end{pmatrix} \left( inv^a \circ g \circ inv^b \right) \right) \circ inv^b = \alpha_0^a \circ \left( id0] \left( \alpha_0^a \circ g \circ \alpha_0^b \right) \right) \circ inv^b =$$

$$= \alpha_0^a \circ \alpha_1^a \circ id0]g \circ \alpha_1^b \circ \alpha_0^b = L_0(0, a, b) \circ id0]g \circ R_0(b)$$

from which (13) follows.

Let $h_1 = \begin{pmatrix} 1 \\ a \\ b \end{pmatrix}$. Then

$$\begin{pmatrix} 1 \\ a \\ b \end{pmatrix} g = \begin{pmatrix} 1 \\ a \\ b \end{pmatrix} id \circ \begin{pmatrix} 0 \\ b \\ b \end{pmatrix} g =$$

$$= inv^a \circ \left( inv^a \circ \begin{pmatrix} 1 \\ a \\ b \end{pmatrix} id \circ inv^b \right) \circ inv^b \circ inv^b \circ \left( inv^b \circ \begin{pmatrix} 0 \\ b \\ b \end{pmatrix} g \circ inv^b \right) \circ inv^b =$$

$$= inv^a \circ \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \left( inv^a \circ id \circ inv^b \right) \circ \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \left( inv^b \circ g \circ inv^b \right) \circ inv^b =$$

$$= \alpha_0^a \circ \alpha 0]inv^{a+b} \circ \alpha_1^b \circ id0]g \circ \alpha_1^b \circ \alpha_0^b = L_0(1, a, b) \circ id0]g \circ R_0(b)$$

from which (13) follows. □

The form (13) is not necessarily minimal. To reduce the number of its elements we can remove from it fragments of the form $\alpha_i \circ \alpha_i$ being equal to $id$.

# Conclusions

In this article we describe groups defined by automata without branches over two-symbol alphabets. Study of semigroups defined by automata without branches

is reduced to that of vectors over finite full transformation semigroups. We also study algebraic properties of the transformations defined by slow-moving automata of finite type. We prove that such invertible transformations can be expressed as compositions of members of the family $\{\alpha_i\}$. In the general case, any slow-moving transformation of finite type can be expressed as a composition of $\alpha_i$, $\beta_i$, $\gamma_i$, $\delta_i$. Further we need to investigate properties of these transformation families and find all relations between these transformations.

# References

[1] B. Csákány, F. Gécseg, *On the groups of automaton permutations*, Kibernetika (Kiev), 1965, No 5, pp. 14–17. (in Russian)

[2] F. Gécseg, *On the groups of one-to-one mappings defined by finite automata*, Kibernetika, Kiev, 1965, No 1, pp. 37–39. (in Russian)

[3] J. Hořejš, *Transformations defined by finite automata*, Probl. kibernetiki, **9** (1963), 23–26. (in Russian)

[4] V. P. Zarovnyi, *Automatonic permutations and group interlacings*, Kibernetika, Kiev, 1965, No. 1, pp. 29–36. (in Russian)

[5] Rostislav I. Grigorchuk, Volodimir V. Nekrashevich, and Vitaliy I. Sushchansky, *Automata, dynamical systems, and groups*, Proceedings of the Steklov Institute of Mathematics, **231** (2000), 128-203.

[6] S.V. Aleshin, *Free semigroup of automata*, Vest. Mosk. Univer. Ser. 1. matem., meh, 1983, 4, pp. 12–14. (in Russian)

[7] A.S. Oliynyk, *On free semigroups of automaton transformations*, Matem. zam, 1998, 63, 2, pp. 248–259. (in Russian)

[8] A.S. Oliynyk, I.I. Reznykov, V.I. Sushchansky, *Transformation semigroups defined by Mealy automata over finite alphabet*, Algerbraic structures and their application: Works of Ukrainian mathematical congress, 2001, pp. 80–99. (in Russian)

[9] I.I. Reznykov, *Mealy automata with two states over two-symbol alphabet, which define finite transformation semigroups*, Visn. Kiyv. Univer. Ser. fiz.-mat. nauk, 2001, 4, pp. 78–86. (in Ukrainian)

[10] I.I. Reznykov, V.I. Sushchansky, *Growth functions of automata with two states over two-symbol alphabet*, Dop. NAN Ukraine, 2002, 2, pp. 76–81. (in Russian)

[11] George H. Mealy, *A method for synthesizing sequential circuits*, Bell System Tech. J. **34**, (1955), 1045–1079

[12] V. M. Glushkov, *The abstract theory of automata*, Russ. Math. Surv., 1961, 16 (5), 1-53.

[13] Ferenc Gécseg, Products of automata, EATCS Monographs on Theoretical Computer Science, 7, Springer-Verlag, Berlin, 1986, viii+107 p.

[14] A.S. Antonenko, E.L. Berkovich, On some algebraic properties of Mealy automata in: Kalmar Workshop on Logic and Computer Science, Szeged, 2003, pp. 59–68.

Acta Cybernetica 18 (2007) 47–60.

# A Classification Scheme for Bin Packing Theory

Edward G. Coffman, Jr.* and János Csirik†

**Abstract**

Classifications of published research place new results in a historical context and in so doing identify open problems. An example in wide use classifies results in scheduling theory according to a scheme originated by Graham, Lawler, Lenstra and Rinnooy Kan [10]. A similar effort was made by Dyckhoff [6] for cutting and packing problems. Such classification schemes can be combined with comprehensive bibliographies, e.g., the one provided for scheduling theory by Bruckner[1]. This paper describes a novel classification scheme for bin packing which is being applied by the authors to an extensive (and growing) bibliography of the theory. Problem classifications are supplemented by compact descriptions of the main results and of the corresponding algorithms. The usefulness of the scheme is extended by an online search engine. With the help of this software, one is easily able to determine whether results already exist for applications that appear to be new, and to assist in locating the cutting edge of the general theory.

## 1 Introduction

For given positive reals $a_1, \ldots, a_n$ and $b_1, b_2, \ldots$, classical bin packing algorithms partition some subset of $\{a_1, \ldots, a_n\}$ into blocks $B_1, B_2, \ldots, B_j$ such that the *levels* $\ell(B_i) := \sum_{a_k \in B_i} a_k$ satisfy the sum constraints $\ell(B_i) \leq b_i$, $1 \leq i \leq j$. This definition embraces several packing problems, depending on the way the subset of the $a_i$'s and the integer $j$ are chosen. In bin packing terms, the $a_i$ are called *items*, the blocks $B_i$ are called *bins* with respective *capacities* or *sizes* $b_i$, and the partitions are called *packings*; the notion of packing items into a sequence of initially empty bins helps visualize algorithms for constructing partitions. It is also helpful in classifying algorithms according to the various constraints under which they must operate in practice. The items are normally given in the form of a sequence or *list* $L = (a_1, \ldots, a_n)$, although the ordering in many cases will not have any significance. To economize on notation, we adopt the harmless abuse whereby $a_i$ denotes both the name and the size of the $i$-th item. The generic symbol for packing is $\mathcal{P}$; the

*Department of Electrical Engineering, Columbia University, 1312 S.W. Mudd, 500 West 120th Street, New York, NY 10027, USA. E-mail: egc@ee.columbia.edu

†Department of Computer Science, University of Szeged, Árpád tér 2, H-6720 Szeged, Hungary. E-mail: csirik@inf.u-szeged.hu

[1]Available at http://www.mathematik.uni-osnabrueck.de/research/OR/class/

number of items in $\mathcal{P}$ is denoted by $|\mathcal{P}|$ and the norm of the packing is defined as the sum of the sizes of the nonempty bins: $\|\mathcal{P}\| := \sum_{\ell(B_i)>0} b_i$. In the majority of problems being classified, the entire list is packed, so $|\mathcal{P}| = n$ and the index $j$ of the last occupied bin is the packing measure of interest. It is also common to have all bin sizes the same, in which case the bin size is denoted simply by $b$ and $a_i \leq b$ is assumed for all $i$. Further, when $b$ functions only as a scale factor, it is usually normalized to 1; *in this case, the norm reduces simply to the number of bins in the packing*, i.e., $j = \|\mathcal{P}\|$. The term *wasted space* has the obvious meaning, $\|\mathcal{P}\| - \sum_{i=1}^{n} a_i$.

With bin sizes given by context, let $\mathcal{P}_A(L)$ denote the packing of $L$ produced by algorithm $A$. In the literature, one finds the notation $A(L)$ representing properties such as $\|\mathcal{P}_A(L)\|$; but since $A(L)$ may denote different properties for different problems (the same algorithm $A$ may apply to problems with different objective functions), we will need the alternative notation on occasion. The more general forms with $b$ specified are $\mathcal{P}(L, b)$ and $A(L, b)$, but the bin size will be omitted whenever it has been normalized to 1. The minimum of $\|\mathcal{P}(L)\|$ over all partitions of $L$ satisfying the sum constraints will have the notation: $OPT(L) := \min_{\mathcal{P}} \|\mathcal{P}(L)\|$, the notation $OPT(L)$ suffering from the same ambiguity as before, i.e., the objective function to which it applies is determined by context. Moreover, in contrast with other algorithm notation, $OPT$ does not denote a unique algorithm.

The classical theory refers to the study of algorithms satisfying various operating constraints which try to minimize, usually only approximately, the number of bins $\|\mathcal{P}(L)\|$ under the sum constraints $\ell(B_i) \leq 1$. *Dual bin packing* changes the sum constraints to $\ell(B_i) \geq 1$ and asks for a packing which *maximizes* the number of bins under these new constraints. Dual bin packing is often called bin covering, a term that we will use here. These combinatorial optimization problems are NP-hard; with problems defined on restricted item sizes or number of items per bin being the major exceptions, this will be the case for nearly all problems in the classifications below.

To fix ideas, consider the Next Fit (NF), First Fit (FF), and Best Fit (BF) approximation algorithms for classical bin packing. Each algorithm packs all the items of $L$ one by one in the sequence $a_1, a_2, \ldots, a_n$. NF packs items in $B_1$ until it encounters an item, say $a_i$, for which $a_i > 1 - \sum_{1 \leq j < i} a_j$; that is, $a_i$ does not fit in the space left over by $a_1, \ldots, a_{i-1}$. At that point $B_1$ is *closed* in the sense that no further items can be packed in $B_1$, and $a_i$ is placed as the first item in $B_2$. This bin-by-bin process repeats, packing the items $a_i, a_{i+1}, \ldots, a_n$ into $B_2, B_3, \ldots$, and continues until no items remain to be packed. The bin being packed at any given step is called the *open* bin. Under FF and BF all bins remain open throughout the packing process. At the $i$-th step under FF, $a_i$ is packed in the lowest indexed bin with sufficient space (of course, this may have to be the empty bin just beyond the last nonempty bin). At the $i$-th step under BF, $a_i$ is packed into a bin in which it fits best, i.e., with the least space left over. In case two or more bins satisfy this criterion, the lowest indexed of these bins is chosen.

Another dual of classical bin packing, called *multiprocessor* or *makespan scheduling*, takes the number, $m$, of bins to be constant and minimizes the capacity $b$ such

that $L$ can be packed into $m$ bins of capacity $b$; again, the norm $\|\mathcal{P}(L)\| = mb$ is minimized, but in this case via $b$ for fixed $m$. List scheduling (LS) is a classical algorithm for this problem and is organized like WORST FIT (a misnomer in the makespan context): the next item to be packed is put in a least-full bin, with ties resolved in favor of lower indexed bins.

Problems fixing the number of bins fall within scheduling theory whose origins in fact predate those of bin packing theory. In scheduling theory, which is very large in its own right, makespan scheduling is more likely to be described as scheduling a list of tasks or jobs (items) on $m$ identical processors (bins) so as to minimize the schedule length or makespan (bin capacity). Our incursion into scheduling problems will be limited to the most elementary duals and applications of bin packing problems, such as the one above.

The most common approach to the analysis of approximation algorithms has been *worst-case* analysis by which the worst possible performance of an algorithm is compared with optimal performance. (Detailed definitions will be provided shortly.) The term *performance guarantee* puts a more positive slant on results of this type[2] . Probability models also enjoy wide use, and are growing in popularity, as they bring out typical, *average-case* behavior rather than the, normally quite rare, worst-case behavior. In probabilistic or stochastic analysis, algorithms have random inputs; the items are usually assumed to be independent, identically distributed random variables. For a given algorithm $A$, $A(L_n)$ is a random variable whose distribution becomes the goal of the analysis. Because of the difficulties inherent to these problems, even for elementary algorithms, one must often settle for weaker results, such as bounds on tail probabilities and asymptotic (large-$n$) estimates of expected values.

An analysis combining aspects of both the combinatorial and probabilistic approaches is that of *stochastic bin packing*, in which a typical problem is to find a packing algorithm that optimizes the expected value of some performance measure. These problems are almost always substantially more difficult extensions of problems that are already quite difficult. The classification scheme will have very few opportunities to cite such results.

The scheme for classifying problems and solutions will take the form of five fields: arena, objective function, class of algorithms, results, and constraints. The arena field describes the nature of the bins[3], such as whether they have variable capacities; the objective function to be minimized or maximized under sum constraints refers to the number of bins of fixed capacity, the capacity of a fixed number of bins, etc.; the class of algorithms refers to paradigms such as online, offline, bounded space, etc. as well as to algorithmic approaches such as *grouping* and *fitting*, to be described in Section 3; the results field specifies performance in terms of absolute or asymptotic worst case ratios, problem complexity, etc.; and constraints refer

[2]So also does the term *competitive analysis*, which usually refers to a worst case analysis comparing an on-line approximation algorithm with an optimal offline algorithm.

[3]The present classification of one dimensional problems will eventually be extended to higher dimensions, in which case the arena field will also specify problem dimensionality (e.g., packing 2-dimensional bins and strip packing.

to limitations in problem parameters, such as a minimum placed on item sizes, a restriction of all data to be integers, and so on. The classification scheme is intended for general use as a compact means for referring to packing problems; however, in the entries of the bibliography, the classification will be supplemented by a brief description of the algorithms studied and the results (typically, but not always, bounds of some kind) derived for the algorithms.

In what follows, Section 2 covers typical results and performance measures, Section 3 describes fundamental algorithms, and then Section 4 contains the details of the classification scheme. The many annotated examples in Section 5 are meant to familiarize the reader with classification criteria and their limitations.

An updated, classified bibliography with a search engine will be available at http://www.inf.u-szeged.hu/~csirik.

## 2 Results

There are many forms results take, but the most common in combinatorial analysis are *performance ratios* or *guarantees*, which give the performance of an approximation algorithm relative to an optimal algorithm. Hereafter, dependence of performance ratios on $\alpha$ means that all item sizes satisfy $a_i \leq \alpha$; this dependence is omitted if there is no upper bound on item size, i.e. $\alpha = b$. For classical bin packing, the *asymptotic* worst-case ratio (or bound) for algorithm $A$ is defined as

$$R_A^{\infty}(\alpha) := \limsup_{k \to \infty} R_A^{(k)}(\alpha)$$

with

$$R_A^{(k)}(\alpha) := \sup_{L:OPT(L)=k} \left\{ \frac{A(L)}{k} \right\}$$

where $OPT(L)$ refers to the optimal offline result. A less formal but more instructive definition describes $R_A^{\infty}(\alpha)$ as the smallest multiplicative constant such that for some additive constant $K < \infty$,

$$A(L) \leq R_A^{\infty}(\alpha) \cdot OPT(L) + K$$

for all $L$.

The *absolute* worst-case ratio is simply

$$R_A(\alpha) := \sup_L \left\{ \frac{A(L)}{OPT(L)} \right\}.$$

The comparison of algorithms by asymptotic bounds can be strikingly different from that by absolute bounds. Generally speaking, the number of items $n$ must be sufficiently large (how large will depend on the algorithm) for the asymptotic bounds to be the better measure for purposes of comparison. Note that the ratios are bounded below by 1; the better algorithms have the smaller ratios.

The performance guarantees for covering have a complementary form. The asymptotic ratio is

$$R_A^\infty := \liminf_{k\to\infty} R_A(k)$$

where

$$R_A(k) := \inf_{L:OPT(L)=k} \left\{ \frac{A(L)}{k} \right\}$$

and the absolute ratio is

$$R_A := \inf_L \left\{ \frac{A(L)}{OPT(L)} \right\}$$

Note that the covering ratios are bounded above by 1; the better algorithms have the larger ratios.

Similar performance guarantees are defined for scheduling and a number of other problems. As can be seen, the ratio notation above is generic; the context will determine which definition is in force. When all item sizes are at most the item size parameter $\alpha$, these bounds are denoted by $R_A^\infty(\alpha)$ and $R_A(\alpha)$.

The determination of time complexities of fundamental algorithms and their extensions or adaptations is usually routine. The analysis of parallel algorithms for computing packings is an example where deriving time complexities is not routine. However, the research in this area, in which results take the form of complexity measures, has been very limited.

Several results quantify the trade-off between the running time of algorithms and the quality of the packings. They produce Polynomial Time (or Fully Polynomial Time) Approximation Schemes [9], denoted by PTAS (or FPTAS). In simplified terms, a typical form of such results is illustrated by: "Algorithm A produces packings with $O(\epsilon)$ expected wasted space and has a running time polynomial in $1/\epsilon$."

Average-case results may be in the form of expected ratios like $\mathbf{E}R_A(L)$ or simply expected performance $\mathbf{E}A(L)$, usually in terms of $\mathbf{E}OPT(L)$. (These comparisons need not be the same of course.) In many cases, tails of the distributions are estimated in the process of deriving estimates for expected values.

# 3 Fundamental algorithms

A number of such algorithms will be incorporated directly into the classification notation. These include the FIT algorithms FF, BF, and WF which we have already described. In some cases only the algorithmic approach or structure will be described, with extensive details omitted. The four structures most often used in defining algorithms are described below.

## 3.1 Fitting algorithms

These refer not only to those just mentioned, but also their offline decreasing counterparts denoted by NFD, FFD, BFD, and WFD, where the D stands for decreasing.

In each case, the algorithm begins with an ordering of $L$ by decreasing item size. The respective algorithms are then applied to the reordered list. The notation for the corresponding *increasing* counterparts simply replaces the D by an I.

Bounded-space algorithms are a subcategory of fitting algorithms and are specified in many cases by a fitting rule, either FF or BF, and a closing rule. The closing rule is invoked when the next item to be packed does not fit into any of the open bins, in which case one of the open bins must be closed and a new bin opened. The choices for the bin to close are the lowest indexed bin (the First bin) and a bin with the highest level (a Best bin).

## 3.2 Grouping algorithms

Grouping is a standard technique that has been studied at great length with many variations. Essentially, it refers to schemes that pack/schedule items based on group membership, where groups are defined by item size. A primary example called HARMONIC and denoted by $H_k$ is based on a partition of the interval $(0, 1]$ into $k$ subintervals, where the partitioning points are $1/2, 1/3, \ldots, 1/k$. Each of these subintervals corresponds to a different group, and each has its own open bin; items belonging to a given group/subinterval are packed only into the corresponding open bin. If a new item arrives that does not fit into the open bin of its group, the bin is closed and a new bin of that type is opened. Thus, the packing of items in each group is an NF packing. Grouping has been defined on other than the $H_k$ intervals, and it has been combined with various greedy fitting algorithms. HARMONIC has received so much coverage in the literature that we adopt, along with the FIT acronyms, the symbol $H_k$ as part of the notation.

## 3.3 Iterative algorithms

Iterating an algorithm designed for good performance under one objective function may be an effective algorithm under another objective function. For example, consider approximation algorithms for the problem of minimizing schedule makespans. One could iterate BFD on an increasing sequence of "candidate" makespans (bin capacities) until one is tried with success, which then yields the desired approximation of the minimum makespan. For iterative versions of fundamental algorithms we use the prefix I. Thus, the algorithm just mentioned would be called IBFD. In a similar approach, IWFD could be used as an approximation algorithm for classical bin packing.

## 3.4 Limiting item sizes or the number packed per bin

If the number of distinct item sizes is limited, to $N$ say, then when $N$ is relatively small, substantial improvements in algorithm design and performance are possible. Moreover, finite (if not actually small) $N$ loses no generality in practice. This assumption leads naturally to integer-program formulations. For example, consider

classical bin packing and define a *configuration* as any subset of items (with replication allowed) with a total size at most 1. Let $C_{jk}$ be the number of items of the $j$-th size in the $k$-th configuration, and let $t_k \equiv t_k(\mathcal{P})$ be the number of bins of $\mathcal{P}$ with the $k$-th configuration. If there is a total of $M$ possible configurations, and if there are $m_j$ items of the $j$-th size in an instance $I$ of the bin packing problem, then finding the size (norm) of an optimal packing is solving the following integer program for $I$ : *minimize* $\sum_{k=1}^{M} t_k$ *subject to* $\sum_{k=1}^{M} t_k C_{jk} = m_j,\ j = 1, \ldots, N,$ *and* $t_k \geq 0,\ k = 1, \ldots M.$

Limiting the number of items per bin is a similar restriction, one that has often been used to greatly simplify average-case analysis. For example, for classical bin packing, there are simple algorithms packing at most 2 items per bin which yield smallest possible asymptotic estimates of expected wasted space, when item sizes are drawn independently and uniformly at random from $[0, 1]$.

# 4 Classification scheme

The notation takes the form

arena|objective_function|algorithm_class|results|constraints

This section gives the current lists of entries for each field, with definitions where needed. The special terms or abbreviations adopted for entries will be given in bold face.

## 4.1 Arena

The basic arena as a sequence of one-dimensional bins has already been described. When sum constraints apply, and all bins have the same size $b$, then the arena field will be empty. When this field is not empty, terms like the following will appear.

1. **variable** $b_i$ means that there is more than one bin size and that there is an unlimited supply for each size.

2. **open_end** refers to problems in which sum constraints are relaxed as follows: bin $B_i$ can always accommodate an item if $\ell(B_i) < b_i$ but it is closed as soon as $\ell(B_i) \geq b_i$. Other notions of exceeding bin capacity will fall under the general term **overfill**.

## 4.2 Objective function

This function will most often be implicit in a term adopted for the corresponding combinatorial optimization problem.

1. **pack** refers to the classical problem of minimizing $\|\mathcal{P}(L)\|$ subject to the sum constraints $\ell(B_i) \leq 1$.

2. **makespan** refers to the problem of minimizing the common bin capacity needed to pack $L$ into a given number of bins. The *bin-stretching* problem is a special case of the makespan problem in which the value of the bin size in the optimal packing is known in advance. For this problem, the term *stretch* will be appended to the performance-guarantee notation.

3. **deadline** abbreviates *deadline scheduling* and refers to the problem of finding schedules in which a maximum cardinality subset of the tasks in $L$ finish by a given deadline (capacity) $b$ on a given number $m$ of processors.

4. **pack_cover** refers to the dual bin packing problem of maximizing $\|\mathcal{P}(L)\|$ subject to the dual constraints $\ell(B_i) \geq 1$.

5. **schedule_cover** refers to the dual makespan scheduling problem of maximizing the makespan $b$ for fixed $m$ such that $\ell(B_i) \geq b$.

   In principle, there are covering versions of deadline scheduling as well, but we have encountered no research on these problems. One such problem is:

6. **deadline_cover** names the problem of minimizing the total size of the subset of tasks needed to cover a given number $m$ of processors with a given deadline $b$.

## 4.3 Algorithm class

1. **offline** algorithms have no constraints beyond the intrinsic sum constraints; an offline algorithm simply maps the entire list $L$ into a packing $\mathcal{P}(L)$. Effectively, all items are known in advance, so the ordering of $L$ plays no role.

2. **online** algorithms sequentially assign items to bins, in the order encountered in $L$, without knowledge of items not yet packed. Thus, the bin to which $a_i$ is assigned is a function only of $a_1, \ldots, a_i$. Note that NF, FF, and BF are all online.

3. **bounded space** algorithms decide where an item is to be packed based only on the current contents of at most a finite number $k$ of bins, where $k$ is a parameter of the algorithm. Note that FF and BF are not bounded space algorithms, but NF is, with $k = 1$. A more precise definition and further discussion of these algorithms appear later.

4. **linear-time** algorithms have $O(n)$ running time. In fact, a more precise statement can be made: all such algorithms classified here take constant time to pack each item. NF is clearly a linear-time algorithm, but FF and BF are not.

   The three characterizations above are orthogonal. But the literature suggests that the following convention will allow us to use one term in classifying algorithms most of the time: *Bounded space implies linear time and linear time implies online.* Exceptions will be noted explicitly; below (under **repack**) we will see how offline algorithms can be linear time.

5. **greedy.** Any algorithm in a broad class of algorithms variously called reasonable, fair, any-fit, or greedy is required to pack the current item into an open bin with sufficient space, in case such a bin exists; in particular, it can not choose to open a new bin in this case. We use the term greedy exclusively to describe such algorithms. Scheduling algorithms satisfying a similar constraint are sometimes called *conservative* or *work conserving.*

6. **repack.** There have been a number of studies devoted to packing problems which allow the repacking (possibly limited in some way) of items – moving an item, say $a_i$, from one bin to another based on the sizes of items $a_j$, $j > i$.

7. **dynamic** packing introduces the time dimension; an instance $L$ of this problem consists of a sequence of triples $(a_i, r_i, d_i)$ with $r_i$ and $d_i$ denoting arrival and departure times, respectively. Under packing algorithm $A$, $A(L,t)$ denotes the number of bins occupied at time $t$, i.e. the number of bins occupied by those items $a_i$ for which $r_i < t < d_i$.

**Conventions:** Along with the algorithm class, the algorithm will be specified when possible. In many cases, the algorithm will be an adaptation or variant of some well-known algorithm, like FF for example, in which case the specification will have the form **FF variant**.

## 4.4 Results

Almost all results fall into the broad classes mentioned in Section 2.

1. Asymptotic worst case ratios, where $\mathbf{R}_{\mathbf{A}}^{\infty}$ is the general entry with $A$ specified where appropriate.

2. Absolute worst case, with $\mathbf{R}_{\mathbf{A}}$ being the entry.

3. Average case: A probabilistic analysis, usually leading only to expected values, is indicated. The entries adopt standard notation such as $\mathbf{E}A(L_n)$ or $\Pr\{A(L_n) > x\}$. In parentheses, a distribution or class of distributions will be given. Examples include $U(0,\alpha)$ (the uniform distribution on $[0,\alpha]$) and $\Delta(0,\alpha)$ (the triangular distribution on $[0,\alpha]$, but if no distribution is specified then it is assumed to be general. Standard terms like **unimodal** and **decreasing** (referring to a density function), etc. will be encountered. Item sizes are assumed to be independent random variables in all cases, unless stated otherwise.

4. Where possible, complexity of the problem will be given in the standard notation of problem complexity.

5. Complexity of the algorithm refers to running-time complexity and will be signalled by the entry **running-time**.

**Conventions:** A paper classified as a worst-case analysis may also have complexity results (but not conversely, unless both types of results figure prominently in the paper, in which case both classifications will be given), and a paper given an average case classification may also have worst case results; here also, both classifications will be noted only if both worst-case and average-case analysis play major roles in the paper. Approximation schemes are classified as complexity results and have entries like **PTAS, FPTAS** as noted earlier.

## 4.5 Constraints

These typically introduce further limitations on the problem instance, or further properties of the algorithm classification.

1. **mutex** stands for mutual exclusion and introduces constraints in the form of a sequence of pairs $(a_i, a_j)$, $i \neq j$, signifying that $a_i$ and $a_j$ can not be put in the same bin.

2. **items/bin** $\leq k$ gives a bound on the number of items that can be packed in a bin.

3. $a_i \leq \alpha$ **or** $a_i \geq \alpha$. These denote bounds on item sizes, the former being far more common in the literature. In the former case $\alpha$ is usually part of the result notation (see the next subsection) so in these cases, it is omitted from the classification. *Throughout the bibliography, the symbol $\alpha$ is reserved for this purpose.* These cases are often called parametric cases in the literature.

   A constraint that may refer as much to analysis as to algorithm design calls for discrete sets of items; as such it is not a significant practical constraint.

4. **discrete** which means that item sizes are all multiples of $1/k$ with $b = 1$. Equivalently, the bin size could be taken as some integer $b$ and item sizes restricted to the set $\{1, \ldots, b\}$.

5. **restricted sizes** refers to the problem where the number of different item sizes is finite.

6. The symbol * refers to features or properties not classifiable within the scheme.

**Conventions:** There are further interesting extensions which occur only in a few papers. In these cases we will use a special notation; a short description in each case will be given as a remark.

# 5 Examples

The following examples should help familiarize the reader with the classification technique.

1. Reference [4] gives an average-case analysis of the classical, bounded-space Next Fit algorithm for bin packing:

   $$|\text{pack}|\text{bounded\_space}|\mathbf{E}NF(L_n), U(0,1)$$

   **Result:** $\mathbf{E}NF(L) = \frac{4}{3}\mathbf{E}OPT(L) + O(1)$, where item sizes are independent draws from U(0,1).

   Recall that the empty arena component implies that all bin levels are bounded above by 1, the common bin size.

2. The classification of [13] shows a nonempty arena field:

   $$\text{variable } b_i|\text{pack}|\text{offline}|\ \text{PTAS}$$

3. The classification of [12] gives another such example:

   $$\text{open\_end}|\text{pack}|\text{online; offline}|R_A^\infty \text{ bound; FPTAS}$$

   **Results:** For the open-end bin packing problem any online algorithm must have an asymptotic worst-case ratio of at least 2. Next Fit achieves this ratio. There is a fully polynomial approximation scheme for this problem.

4. The classification of [2] illustrates a makespan problem:

   $$|\text{makespan}|\text{online}|R_A \text{ stretch}$$

   **Results:** A combined algorithm achieves a worst case bound of 1.625. The best lower bound for any online algorithm is 4/3.

5. Reference [3] shows a deadline objective function:

   $$|\text{deadline}|\text{offline; WFI, FFI}|R_A$$

   **Results:** $R_{WFI} = \frac{1}{2}, R_{FFI} = \frac{3}{4}$.

6. Reference [5] is classified as a covering problem.

   $$|\text{pack\_cover}|\text{online}|R_A^\infty \text{ bound}$$

   **Results:** Asymptotic bound: $R_A^\infty \leq 1/2$ for any online algorithm $A$. There exists an asymptotically optimal online algorithm.

7. The classification of [7] is

$$|\text{pack}|\text{offline; combined BF, FFD variant}|R_A^\infty$$

**Algorithm:** Combined Best Fit (CBF) which takes the better of the First Fit Decreasing solution and Best Two Fit (B2F) solution, where the latter algorithm is a grouping version of Best Fit limiting the number of items per bin.

**Results:**

$R_{B2F}^\infty = 5/4, \quad \frac{227}{195} \leq R_{CBF}^\infty \leq \frac{6}{5}$

Note that the word 'variant' may be simplistic in that it occasionally hides details of relatively complicated algorithms.

8. Reference [8] shows an example for combination of two algorithm classes:

$$|\text{pack}|\text{bounded\_space, repack}|R_A$$

**Algorithm:** $\text{REP}_3$: an adaptation of FFD using three open bins at any time.

**Result:** $R_{REP_3}^\infty \approx 1.69\ldots$ .

9. The classification of [11] illustrates a constraint:

$$|\text{pack}|\text{offline; FF variant}|R_A^\infty|\text{items/bin} \leq k$$

**Result:**

$$\left(\frac{27}{10} - \left\lceil \frac{37}{10k} \right\rceil\right) \leq R_{FF_k}^\infty \leq \left(\frac{27}{10} - \frac{24}{10k}\right),$$

where $\text{FF}_k$ is the obvious adaptation of FF.

10. For [14], the classification mentions yet another constraint:

$$|\text{pack}|\text{bounded\_space; } \text{H}_k \text{ variant}|R_A^\infty|O(\log k) \text{ open bins}$$

**Result:** $R_{SH_k}^\infty(k) = R_{H_k}^\infty$, where $\text{SH}_k$ is a simplified version of $\text{H}_k$ that uses only $O(\log k)$ open bins at any time.

11. Classification of [1] aggregates several results:

$$|\text{pack\_cover}|\text{offline, online; NF, FFD, IWFD variants}|R_A^\infty$$

**Algorithms:** Adaptation of Next Fit called DNF (a new bin is opened when the current open bin, say $B$, first overflows with item, say $a_i$, but in this case $a_i$ stays in $B$); of First Fit Decreasing with a parameter $r$ ($\text{FFD}_r$); and of an iterated version of Worst Fit (IWFD).

**Results:**

$$R_{DNF}^\infty = \frac{1}{2}, FFD_r^\infty = \frac{2}{3} \text{ for all } r, \frac{4}{3} \leq r \leq \frac{3}{2}, \text{ and } R_{IWFD}^\infty = \frac{3}{4}.$$

# References

[1] S. B. Assman, D. S. Johnson, D. J. Kleitman, and J. Y-T. Leung. On a dual version of the one-dimensional bin packing problem. *J. Algorithms*, 5:502–525, 1984.

$$|\text{pack\_cover}|\text{offline, online; NF, FFD, IWFD variants}|R_A^\infty$$

[2] Y. Azar and O. Regev. On-line bin-stretching. *Theor. Comp. Sci.*, 268:17–41, 2001.

$$|\text{makespan}|\text{online}|R_A \text{ stretch}$$

[3] E. G. Coffman, Jr., J. Y. Leung, and D. W. Ting. Bin packing: maximizing the number of pieces packed. *Acta Informatica*, 9:263–271, 1978.

$$|\text{deadline}|\text{offline; WFI, FFI}|R_A$$

[4] E. G. Coffman, Jr., K. So, M. Hofri, and A. C. Yao. A stochastic model of bin-packing. *Inf. and Cont.*, 44:105–115, 1980.

$$|\text{pack}|\text{bounded\_space}|\mathbf{E}NF(L_n), U(0,1)$$

[5] J. Csirik and V. Totik. On-line algorithms for a dual version of bin packing. *Disc. Appl. Math.*, 21:163–167, 1988.

$$|\text{pack\_cover}|\text{online}|R_A^\infty \text{ bound}$$

[6] H. Dyckhoff. A typology of cutting and packing problems. *Eur. J. Oper. Res.*, 44:145–159, 1990.

[7] D. K. Friesen and M. A. Langston. Analysis of a compound bin-packing algorithm. *SIAM J. Disc. Math.*, 4:61–79, 1991.

$$\text{pack}|\text{offline; combined BF, FFD variant}|R_A^\infty$$

[8] G. Galambos and G. J. Woeginger. Repacking helps in bounded space on-line bin-packing. *Computing*, 49:329–338, 1993.

$$|\text{pack}|\text{bounded\_space; repack}|R_A$$

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* W. H. Freeman, New York, New York, 1979.

[10] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals Disc. Math.*, 5:287–326, 1979.

[11] K. L. Krause, Y. Y. Shen, and H. D. Schwetman. Analysis of several task-scheduling algorithms for a model of multiprogramming computer systems. *J. ACM*, 22:522–550, 1975.

$$|\text{pack}|\text{offline; FF variant}|R_A^\infty|\text{items/bin} \leq k$$

[12] J. Y.-T. Leung, M. Dror, and G. H. Young. A note on an open-end bin packing problem. *J. of Scheduling*, 4:201–207, 2001.

$$\text{open\_end}|\text{pack}|\text{online; offline}|R_A^\infty \text{ bound; FPTAS}$$

[13] F. D. Murgolo. An efficient approximation scheme for variable-sized bin packing. *SIAM J. Comput.*, 16:149–161, 1988.

$$\text{variable } b_i|\text{pack}|\text{offline}| \text{ PTAS}$$

[14] G. J. Woeginger. Improved space for bounded-space, on-line bin-packing. *SIAM J. Disc. Math.*, 6:575–581, 1993.

$$|\text{pack}|\text{bounded\_space; } \mathrm{H}_k \text{ variant}|R_A^\infty|O(\log k) \text{ open bins}$$

Acta Cybernetica 18 (2007) 61–75.

# Functional Equations, Constraints, Definability of Function Classes, and Functions of Boolean Variables*

Miguel Couceiro† and Stephan Foldes‡

**Abstract**

The paper deals with classes of functions of several variables defined on an arbitrary set $A$ and taking values in a possibly different set $B$. Definability of function classes by functional equations is shown to be equivalent to definability by relational constraints, generalizing a fact established by Pippenger in the case $A = B = \{0, 1\}$.

Conditions for a class of functions to be definable by constraints of a particular type are given in terms of stability under certain functional compositions. This leads to a correspondence between functional equations with particular algebraic syntax and relational constraints with certain invariance properties with respect to clones of operations on a given set.

When $A = \{0, 1\}$ and $B$ is a commutative ring, such $B$-valued functions of $n$ variables are represented by multilinear polynomials in $n$ indeterminates in $B[X_1, \ldots, X_n]$. Functional equations are given to describe classes of field-valued functions of a specified bounded degree. Classes of Boolean and pseudo-Boolean functions are covered as particular cases.

**Keywords:** Function classes, class composition, stability, functional equations, relational constraints, function class definability, ring-valued functions, multilinear polynomial representations, linear equations, field-valued functions of Boolean variables, Boolean functions, pseudo-Boolean functions.

## 1 Introduction and Basic Definitions

For arbitrary sets $B$ and $C$, by a *$C$-valued function on $B$* we mean a map

$$f : B^n \to C$$

*The work of the first named author was partially supported by the Graduate School in Mathematical Logic MALJA. Supported in part by grant #28139 from the Academy of Finland

†Department of Mathematics, Statistics and Philosophy University of Tampere Kansleririnne 1, 33014 Tampere, Finland E-mail: Miguel.Couceiro@uta.fi

‡Institute of Mathematics, Tampere University of Technology PL553, 33101 Tampere, Finland, E-mail: stephan.foldes@tut.fi

where $n \geq 1$ is called the *arity* of $f$. The *essential arity* of an $n$-ary $C$-valued function $f : B^n \to C$ is defined as the cardinality of the set of indices

$$I = \{1 \leq i \leq n : \text{ there are } a_1, \ldots a_{i-1}, a_i, b_i, a_{i+1}, \ldots, a_n \text{ with } a_i \neq b_i \text{ and } f(a_1, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n) \neq f(a_1, \ldots, a_{i-1}, b_i, a_{i+1}, \ldots, a_n)\}.$$

For each $i \in I$, we say that the $i$th *variable of $f$ is essential.* Note that the essential arity of $f$ is zero if and only if $f$ is constant. If $B = C$, then a $C$-valued function on $B$ is called an *operation on $B$*. Operations on the two-element set $B = \{0, 1\}$ are usually refered to as *Boolean functions.*

For any maps $g_1, \ldots, g_n : D \to B$, where $D$ is any set and $f : B^n \to C$, the *composition* $f(g_1, \ldots, g_n)$ is defined as the map from $D$ to $C$ given by $f(g_1, \ldots, g_n)(a) = f(g_1(a), \ldots, g_n(a))$, for every $a \in D$.

Let $A$, $B$ and $C$ be arbitrary non-empty sets, $\mathcal{I}$ a class (i.e. set) of $C$-valued functions on $B$ (of various arities), and $\mathcal{J}$ a class of $B$-valued functions on $A$ (of various arities). The *class composition* $\mathcal{I}\mathcal{J}$ is defined as the set

$$\mathcal{I}\mathcal{J} = \{f(g_1, \ldots, g_n) \mid n, m \geq 1, f \text{ } n\text{-ary in } \mathcal{I}, g_1, \ldots, g_n \text{ } m\text{-ary in } \mathcal{J}\}.$$

If $\mathcal{I}$ is a singleton, $\mathcal{I} = \{f\}$, then we write $f\mathcal{J}$ for $\{f\}\mathcal{J}$. We note that this construction underlies the various notions of subfunction and minor appearing e.g. in [13, 12, 15, 3, 8, 4].

Consider arbitrary non-empty sets $A$, $B$, and $C$, and let $\mathcal{I}$ be a class of $C$-valued functions on $B$ and $\mathcal{J}$ a class of $B$-valued functions on $A$. We say that $\mathcal{I}$ is *stable under right composition with* $\mathcal{J}$ if $\mathcal{I}\mathcal{J} \subseteq \mathcal{I}$. Similarly, we say that $\mathcal{J}$ is *stable under left composition with* $\mathcal{I}$ if $\mathcal{I}\mathcal{J} \subseteq \mathcal{J}$. Note that a clone on an arbitrary set $A$ is simply a class $\mathcal{C}$ of $A$-valued functions on $A$ that contains all projections, and is stable under (left or right) composition with itself, i.e. $\mathcal{C}\mathcal{C} \subseteq \mathcal{C}$ (or equivalently, $\mathcal{C}\mathcal{C} = \mathcal{C}$).

Consider arbitrary non-empty sets $A$ and $B$. A *functional equation* (for $B$-valued function on $A$) is a formal expression

$$h_1(\mathbf{f}(g_1(\mathbf{v}_1, \ldots, \mathbf{v}_p)), \ldots, \mathbf{f}(g_m(\mathbf{v}_1, \ldots, \mathbf{v}_p))) = \\ = h_2(\mathbf{f}(g'_1(\mathbf{v}_1, \ldots, \mathbf{v}_p)), \ldots, \mathbf{f}(g'_t(\mathbf{v}_1, \ldots, \mathbf{v}_p))) \quad (1)$$

where $m, t, p \geq 1$, $h_1 : B^m \to C$, $h_2 : B^t \to C$, each $g_i$ and $g'_j$ is a map $A^p \to A$, the $\mathbf{v}_1, \ldots, \mathbf{v}_p$ are $p$ distinct symbols called *vector variables*, and $\mathbf{f}$ is a distinct symbol called *function symbol.*

For $n \geq 1$, we denote by $\mathbf{n}$ the set $\mathbf{n} = \{1, \ldots, n\}$, so that an $n$-vector ($n$-tuple) $v$ in $A^n$ is a map $v : \mathbf{n} \to A$. In this way, if $g$ is an $p$-ary operation on $A$ and $v_1, \ldots, v_p$ are $n$-vectors in $A^n$, then $g(v_1, \ldots, v_p)$ denotes the $n$-vector

$$(g(v_1, \ldots, v_p)(1), \ldots, g(v_1, \ldots, v_p)(n)) \in A^n.$$

For an $n$-ary $B$-valued function on $A$, $f : A^n \to B$, we say that $f$ *satisfies* the

equation (1) if, for all $v_1, \ldots, v_p \in A^n$, we have

$$h_1(f(g_1(v_1, \ldots, v_p)), \ldots, f(g_m(v_1, \ldots, v_p))) = \\ = h_2(f(g'_1(v_1, \ldots, v_p)), \ldots, f(g'_t(v_1, \ldots, v_p))). \quad (2)$$

A class (i.e. set) $\mathcal{K}$ of $B$-valued functions on $A$ is said to be *defined*, or *definable*, by a set $\mathcal{E}$ of functional equations, if $\mathcal{K}$ is the class of all those functions which satisfy every member of $\mathcal{E}$.

To illustrate, let $A = B = \{0,1\}$, $m = 2$, $t = 1$, $p = 2$, and let $g_1$ be the projection function $(x,y) \mapsto x$, $g_2$ the conjunction $(x,y) \mapsto xy$, $h_1 = g_2$, and $h_2$ the identity $x \mapsto x$. The functional equation (1) so specified defines the clone (Post class) of monotone Boolean functions. In a more free style of notation, this equation can be displayed as

$$\mathbf{f(v_1)f(v_1v_2) = f(v_1v_2)}.$$

When the specific context is well understood, we shall present functional equations in such more informal manner.

Useful functional properties have often been advantangeously expressed by functional equations. Classical examples include the linearity of $\mathbb{F}$-valued functions on a field $\mathbb{F}$, as well as monotonicity and convexity properties traditionally expressed by functional inequalities which are obviously equivalent to functional equations in max-plus language. More contemporary examples include the submodular property of real-valued functions $\{0,1\}^n \longrightarrow \mathbb{R}$, and Post classes (clones) of Boolean functions traditionally characterized by relations. Many strong consequences of submodularity, such as the Hall-Rado theorems, follow directly from the characterizing submodular inequality which is essentially a max-plus functional equation (see Welsh [14]). For Boolean functions, equations were systematically studied in [3] and, in a variant form, by Pogosyan [9]. Also, in [5] equations were shown to provide a measure of complexity, essentially in terms of the syntax of the functional equations used to define Post classes.

## 2 Definability of Function Classes by Functional Equations and Relational Constraints

An *$m$-ary relation on $A$* is a subset $R$ of $A^m$, and thus the relation $R$ can be viewed as a class (set) of unary maps from $\mathbf{m}$ to $A$. A function $f : A^n \to A$ is said to *preserve* $R$, and $R$ is said to be *invariant under* $f$, if $fR \subseteq R$, where $fR$ is the class composition $\{f\}R$ as explained above. An $m$-ary *$A$-to-$B$ constraint* (or simply, $m$-ary *constraint*, when the underlying sets are understood from the context) is a couple $(R,S)$ where $R \subseteq A^m$ and $S \subseteq B^m$. The relations $R$ and $S$ are called the *antecedent* and *consequent*, respectively, of the relational constraint (Pippenger [8]). A $B$-valued function on $A$, $f : A^n \longrightarrow B$, $n \geq 1$, is said to *satisfy* an $m$-ary $A$-to-$B$ constraint $(R,S)$ if $fR \subseteq S$. A class $\mathcal{K}$ of $B$-valued functions on $A$ is said to be *defined*, or *definable*, by a set $\mathcal{T}$ of $A$-to-$B$ constraints, if $\mathcal{K}$ is the class of all those functions which satisfy every constraint in $\mathcal{T}$.

As an example, the already mentioned clone of monotone Boolean functions can be equivalently defined by the single constraint $(\leq, \leq)$, where $\leq$ denotes the less-or-equal relation on $\{0,1\}$.

In [8], Pippenger has shown that in the Boolean case, i.e. when $A = B = \{0,1\}$, definability of a function class by functional equations is equivalent to definability by relational constraints. The following theorem is not restricted to the Boolean case, and not even contingent on the finiteness of the underlying sets.

**Theorem 1.** *Let $A$ be an arbitrary non-empty set, and $B$ any set with at least two elements. For any class $\mathcal{K}$ of $B$-valued functions on $A$, the following are equivalent:*

*(i) $\mathcal{K}$ is definable by some set of functional equations;*

*(ii) $\mathcal{K}$ is definable by some set of relational constraints.*

*Proof.* To prove that $(i) \Rightarrow (ii)$, it is enough to show that for every functional equation (1) there is a relational constraint $(R, S)$, such that the $B$-valued functions on $A$ satisfying the equation are exactly the same as those satisfying the constraint. Indeed, we can define the constraint $(R, S)$ by

$$\begin{aligned} R &= \{(g_1(a), \ldots, g_m(a), {g'}_1(a), \ldots, {g'}_t(a)) : a \in A^p\}, \\ S &= \{(b_1, \ldots, b_m, b'_1, \ldots, b'_t) \in B^{m+t} : h_1(b_1, \ldots, b_m) = h_2(b'_1, \ldots, b'_t)\}. \end{aligned}$$

Conversely, let us show that $(ii) \Rightarrow (i)$. Let $\mathcal{T}$ be a set of constraints, and let $\mathcal{K}$ be the class of $B$-valued functions on $A$ defined by $\mathcal{T}$. Consider the set $\mathcal{T}'$ of constraints obtained from $\mathcal{T}$ by removing all those constraints with empty antecedent. Clearly, $\mathcal{T}$ and $\mathcal{T}'$ define the same class $\mathcal{K}$ of $B$-valued functions on $A$. Therefore, the proof will be complete if we can show that for every constraint $(R, S)$ with $R \neq \emptyset$ there is a functional equation (1) satisfied by exactly the same functions as those satisfying $(R, S)$.

Let $m$ be the arity of $(R, S)$. The construction of the equation (1) is based on the following facts.

**Fact 1.** *Given a non-empty relation $R \subseteq A^m$, there is a $p \geq 1$ and a map $g : A^p \to A^m$, such that the range of $g$ is $R$.*

**Fact 2.** *Given a relation $S \subseteq B^m$, there exist maps $h_1, h_2 : B^m \to B$, such that*

$$S = \{b \in B^m : h_1(b) = h_2(b)\}.$$

Using these functions $g, h_1$ and $h_2$, the equation (1) can be defined as follows: the integer $m$ is the arity of $(R, S)$, $t = m$, and $p$ is the arity of $g : A^p \to A^m$. For $1 \leq i \leq m = t$, let $g_i = {g'}_i$ be the $i$th component of $g$, i.e. we have

$$g(a) = (g_1(a), \ldots, g_m(a))$$

for all $a \in A^p$. The maps $h_1, h_2$ in (1) are given by Fact 2. □

It is not difficult to see that both Fact 2 and Theorem 1 itself would fail if we allowed $B$ to be a singleton. However, the implication $(i) \Rightarrow (ii)$ in Theorem 1 would continue to hold.

# 3 Definability of Function Classes by Invariant Constraints

The question of definability of Boolean function classes by constraints $(R, S)$, where $R, S \subseteq \{0,1\}^n$ are of a special algebraic kind, was considered in [1]. Specifically, the relations $R$ and $S$ were required to be affine subspaces of the vector space $\{0,1\}^n$ over the two-element field $\mathbf{GF}(2)$. A subset of $\{0,1\}^n$ is an affine subspace if and only if it is closed under the triple sum operation $u+v+w$, i.e. if and only if it is invariant under the clone $\mathcal{L}_{01}$ of constant-preserving linear Boolean functions - that is, functions which are the sum of an odd number of variables. (See e.g. Godement [6].) Also it is well known that the non-empty affine subspaces can be described as ranges of affine maps, and that affine hyperplanes can be described as kernels of affine forms, i.e. as sets on which a given form agrees with the null form. As shown in [1], this accounts for the definability of certain function classes by linear equations.

In this section we consider general notions of closure for the antecedent $R$ and the consequent $S$ of a constraint $(R, S)$, and we address the question of definability of classes of $B$-valued functions on a set $A$ by such invariant constraints, without any restriction on the underlying sets $A$ and $B$.

**Associativity Lemma.** *Consider arbitrary non-empty sets $A$, $B$, $C$ and $E$, and let $\mathcal{I}$ be a class of $E$-valued functions on $C$, $\mathcal{J}$ a class of $C$-valued functions on $B$, and $\mathcal{K}$ a class of $B$-valued functions on $A$. The following hold:*

*(i)* $(\mathcal{I}\mathcal{J})\mathcal{K} \subseteq \mathcal{I}(\mathcal{J}\mathcal{K})$;

*(ii) If $\mathcal{J}$ is stable under right composition with the clone of projections on $B$, then* $(\mathcal{I}\mathcal{J})\mathcal{K} = \mathcal{I}(\mathcal{J}\mathcal{K})$.

*Proof.* The inclusion $(i)$ is a direct consequence of the definition of function class composition. Property $(ii)$ asserts that the converse inclusion also holds if $\mathcal{J}$ is stable under right composition with projections. A typical function in $\mathcal{I}(\mathcal{J}\mathcal{K})$ is of the form

$$f(g_1(h_{11}, \ldots, h_{1m_1}), \ldots, g_n(h_{n1}, \ldots, h_{nm_n}))$$

where $f$ is in $\mathcal{I}$, the $g_i$'s are in $\mathcal{J}$, and the $h_{ij}$'s are in $\mathcal{K}$. By taking appropriate functions $g'_1, \ldots, g'_n$ obtained from $g_1, \ldots, g_n$ by addition of inessential variables and permutation of variables, the function above can be expressed as

$$f(g'_1(h_{11}, \ldots, h_{1m_1}, \ldots, h_{n1}, \ldots, h_{nm_n}), \ldots, g'_n(h_{11}, \ldots, h_{1m_1}, \ldots, h_{n1}, \ldots, h_{nm_n}))$$

which is easily seen to be in $(\mathcal{I}\mathcal{J})\mathcal{K}$ . □

Note that statement $(ii)$ of the Associativity Lemma applies, in particular, if $\mathcal{J}$ is any clone on $C = B$.

Let $\mathcal{F}$ be a set of $B$-valued functions on $A$. If $\mathcal{P}$ is the clone of all projections on $A$, then $\mathcal{F}\mathcal{P} = \mathcal{F}$ expresses closure under taking minors as in [8], or closure under simple variable substitutions in the terminology of [2].

For a class $\mathcal{F}$ of $A$-valued functions on $A$, an $m$-ary relation $R$ on $A$ is said to be *$\mathcal{F}$-invariant* if $\mathcal{F}R \subseteq R$. In other words, $R$ is $\mathcal{F}$-invariant if every member of $\mathcal{F}$ preserves $R$. If two classes of functions $\mathcal{F}$ and $\mathcal{G}$ generate the same clone, then the $\mathcal{F}$-invariant relations are the same as the $\mathcal{G}$-invariant relations. (See Pöschel [10] and [11].) Observe that we always have $R \subseteq \mathcal{F}R$ if $\mathcal{F}$ contains the projections, but we can have $R \subseteq \mathcal{F}R$ even if $\mathcal{F}$ contains no projections. (Take the Boolean triple sum $x_1 + x_2 + x_3$ as the only member of $\mathcal{F}$.)

For a clone $C$, the intersection of $m$-ary $C$-invariant relations is always $C$-invariant and it is easy to see that, for an $m$-ary relation $R$, the smallest $C$-invariant relation containing $R$ in $A^m$ is $CR$, and it is said to be *generated by* $R$. (See [10] and [11], where Pöschel denotes $CR$ by $\Gamma_C(R)$.)

Let $C_1$ and $C_2$ be clones on arbitrary non-empty sets $A$ and $B$, respectively. If $R$ is $C_1$-invariant and $S$ is $C_2$-invariant, we say that $(R, S)$ is a *$(C_1, C_2)$-constraint*. The following result generalizes Lemma 1 in [1]:

**Lemma 2.** *Consider arbitrary non-empty sets $A$ and $B$. Let $f$ be a $B$-valued function on $A$, and let $C$ be a clone on $A$. If every function in $fC$ satisfies an $A$-to-$B$ constraint $(R, S)$, then $f$ satisfies $(CR, S)$.*

*Proof.* The assumption means that $(fC)R \subseteq S$. By the Associativity Lemma, $(fC)R = f(CR)$, and thus $f(CR) \subseteq S$. □

A class $\mathcal{K}$ of $B$-valued functions on $A$ is said to be *locally closed* if for every $B$-valued function $f$ on $A$ the following holds: if every finite restriction of $f$ (i.e restriction to a finite subset) coincides with a finite restriction of some member of $\mathcal{K}$, then $f$ belongs to $\mathcal{K}$.

**Theorem 3.** *Consider arbitrary non-empty sets $A$ and $B$ and let $C_1$ and $C_2$ be clones on $A$ and $B$, respectively. For any class $\mathcal{K}$ of $B$-valued functions on $A$, the following conditions are equivalent:*

*(i) $\mathcal{K}$ is locally closed and it is stable both under right composition with $C_1$ and under left composition with $C_2$;*

*(ii) $\mathcal{K}$ is definable by some set of $(C_1, C_2)$-constraints.*

*Proof.* To show that $(ii) \Rightarrow (i)$, assume that $\mathcal{K}$ is definable by some set $\mathcal{T}$ of $(C_1, C_2)$-constraints. For every $(R, S)$ in $\mathcal{T}$, we have $\mathcal{K}R \subseteq S$. Since $R$ is $C_1$-invariant, $\mathcal{K}R = \mathcal{K}(C_1R)$. By the Associativity Lemma, $\mathcal{K}(C_1R) = (\mathcal{K}C_1)R$, and therefore $(\mathcal{K}C_1)R = \mathcal{K}R \subseteq S$. Since this is true for every $(R, S)$ in $\mathcal{T}$ we must have $\mathcal{K}C_1 \subseteq \mathcal{K}$.

For every $(R, S)$ in $\mathcal{T}$, we have $\mathcal{K}R \subseteq S$, and therefore $C_2(\mathcal{K}R) \subseteq C_2S$. By the Associativity Lemma, $(C_2\mathcal{K})R \subseteq C_2(\mathcal{K}R) \subseteq C_2S$, and $C_2S = S$ because $S$ is $C_2$-invariant. Thus $(C_2\mathcal{K})R \subseteq S$ for every $(R, S)$ in $\mathcal{T}$, and we must have $C_2\mathcal{K} \subseteq \mathcal{K}$.

To see that $\mathcal{K}$ is locally closed, consider $f \notin \mathcal{K}$, say of arity $n \geq 1$, and let $(R, S)$ be an $m$-ary $(C_1, C_2)$-constraint that is satisfied by every function $g$ in $\mathcal{K}$ but not satisfied by $f$. Hence for some $a^1, \ldots, a^n$ in $R$, $f(a^1, \ldots, a^n) \notin S$ but

$g(a^1,\ldots,a^n) \in S$, for every $n$-ary function $g$ in $\mathcal{K}$. Thus the restriction of $f$ to the finite set $\{(a^1(i),\ldots,a^n(i)) : i \in \mathbf{m}\}$ does not coincide with that of any member of $\mathcal{K}$.

To prove $(i) \Rightarrow (ii)$, we show that for every function $g$ not in $\mathcal{K}$, there is a $(\mathcal{C}_1,\mathcal{C}_2)$-constraint $(R,S)$ which is satisfied by every member of $\mathcal{K}$ but not satisfied by $g$. The class $\mathcal{K}$ will then be definable by the set $\mathcal{T}$ of those $(\mathcal{C}_1,\mathcal{C}_2)$-constraints that are satisfied by all members of $\mathcal{K}$.

Note that $\mathcal{K}$ is a fortiori stable under right composition with the clone containing all projections, that is, $\mathcal{K}$ is closed under simple variable substitutions. We may assume that $\mathcal{K}$ is non-empty. Suppose that $g$ is an $n$-ary $B$-valued function on $A$ which is not in $\mathcal{K}$. Since $\mathcal{K}$ is locally closed, there is a finite restriction $g_F$ of $g$ to a finite subset $F \subseteq A^n$ such that $g_F$ disagrees with every function in $\mathcal{K}$ restricted to $F$. Suppose that $F$ has size $m$, and let $a^1,\ldots,a^n$ be $m$-tuples in $A^m$, such that $F = \{(a^1(i),\ldots,a^n(i)) : i \in \mathbf{m}\}$. Define $R_0$ to be the set $\{a^1,\ldots,a^n\}$, and let $S = \{f(a^1,\ldots,a^n) : f \in \mathcal{K}, f \text{ } n\text{-ary}\}$. Clearly, $(R_0,S)$ is not satisfied by $g$, and it is not difficult to see that every member of $\mathcal{K}$ satisfies $(R_0,S)$. As $\mathcal{K}$ is stable under left composition with $\mathcal{C}_2$, it follows that $S$ is $\mathcal{C}_2$-invariant. Let $R$ be the $\mathcal{C}_1$-invariant relation generated by $R_0$, i.e. $R = \mathcal{C}_1 R_0$. By Lemma 2, the constraint $(R,S)$ constitutes indeed the desired separating $(\mathcal{C}_1,\mathcal{C}_2)$-constraint. □

This generalizes the characterizations of closed classes of functions given by Pippenger in [8] as well as in [1] and [2] by considering arbitrary underlying sets, possible infinite, and more general closure conditions. In the finite case, we obtain as special cases of Theorem 3 the characterizations given in Theorem 2.1 and Theorem 3.2 in [8], by taking $\mathcal{C}_1 = \mathcal{C}_2 = \mathcal{P}$, and $\mathcal{C}_1 = \mathcal{U}$ and $\mathcal{C}_2 = \mathcal{P}$, respectively, where $\mathcal{U}$ is a clone containing only functions having at most one essential variable, and $\mathcal{P}$ is the clone of all projections. Taking $A = B = \{0,1\}$ and $\mathcal{C}_1 = \mathcal{C}_2 = \mathcal{L}_{01}$, we obtain the characterization of classes of Boolean functions definable by sets of affine constraints given in [1]. For arbitrary non-empty underlying sets, Theorem 1 in [2] corresponds to the particular case $\mathcal{C}_1 = \mathcal{C}_2 = \mathcal{P}$. In this case, from Theorem 1 and Theorem 3 we conclude the following:

**Corollary 4.** *Consider arbitrary non-empty sets $A$ and $B$. The equationally definable classes of $B$-valued functions on $A$ are exactly those locally closed classes that are stable under right composition with the clone of projections on $A$.*

In certain cases, given a $(\mathcal{C}_1,\mathcal{C}_2)$-constraint $(R,S)$, $R \subseteq A^m$, $S \subseteq B^m$, the construction of a functional equation given in the proof of Theorem 1 in the previous section can be refined to yield a functional equation with special algebraic syntax. To do this, one may seek to use, instead of arbitrary functions as given by Fact 1 and Fact 2 in the proof of Theorem 1, functions $g_1,\ldots,g_m,h_1,h_2$ of a particular kind still satisfying the conditions of these Facts. For example, in [1], the functions were chosen to be affine maps, based on the range-and-kernel theory of linear algebra. Another application of this strategy will be given in Section 4.

Also, in certain cases, given a functional equation (1) with a special algebraic syntax, if the functions $g_1,\ldots,g_m,g'_1,\ldots,g'_t,h_1,h_2$ appearing in the equation have

particular structure-preserving properties, then it may be possible to conclude that the construction of the constraint $(R, S)$, as given in the first part of the proof of Theorem 1, yields relations $R$ and $S$ invariant under certain clones $C_1$ and $C_2$. Thus the affine functions appearing in the "linear" functional equations defined in [1] were used to construct affine constraints. The same principle, together with Theorem 3, will be used in Section 4 to show that certain natural function classes cannot be defined by a particular type of functional equations.

# 4 Functions of Boolean Variables Valued in a Ring

In this section we consider functions $\{0,1\}^n \to B$, where $B$ is a commutative ring. We view $\{0,1\}$ as endowed with the two-element field structure, $\{0,1\} = \mathbf{GF}(2)$, as well as with the lattice structure where $0 < 1$. If $B$ is also $\{0,1\} = \mathbf{GF}(2)$, then these $B$-valued functions are called Boolean functions. If $B$ is the field $\mathbb{R}$ of real numbers, then the functions under consideration are called *pseudo-Boolean functions*, which provide an algebraic representation for set functions $\mathcal{P}(E) \to \mathbb{R}$ for finite $E$ (see e.g. [4] for a recent reference).

Every Boolean function $\{0,1\}^n \to \{0,1\}$ is well known to be representable by a unique multilinear polynomial in $n$ indeterminates over $\mathbf{GF}(2)$, i.e. a polynomial which is linear in each of its indeterminates, called its *Zhegalkin polynomial*, *Reed-Muller polynomial* or *ring-sum expansion*. Also, pseudo-Boolean functions can be uniquely represented by multilinear polynomials in $n$ indeterminates over $\mathbb{R}$ (see Hammer and Rudeanu [7]).

Consider any commutative ring $B$ with null and identity elements $0_B$ and $1_B$, respectively. For a polynomial $p \in B[X_1, \ldots, X_n]$ in $n$ indeterminates, and an $n$-tuple $(a_1, \ldots, a_n) \in \{0,1\}^n$, for each $a_i$ let $a_i^B$ denote $0_B$ or $1_B$ according to whether $a_i$ is 0 or 1, and denote the evaluation $p(a_1^B, \ldots, a_n^B)$ simply by $p(a_1, \ldots, a_n)$. The $B$-valued function on $\{0,1\}$ given by

$$(a_1, \ldots, a_n) \mapsto p(a_1, \ldots, a_n)$$

is said to be *represented* by $p$. By a method similar to that used by Hammer and Rudeanu [7] in the case $B = \mathbb{R}$, we show in the next theorem the existence of a unique multilinear polynomial representation for any $B$-valued function on $\{0,1\}$, for any commutative ring $B$ with identity. This unifies the Zhegalkin and pseudo-Boolean polynomial representations.

**Theorem 5.** *Consider any commutative ring $B$ with identity. For any $n \geq 1$, every $B$-valued function $f$ on $\{0,1\}$, $f : \{0,1\}^n \to B$, is represented by a unique multilinear polynomial $p \in B[X_1, \ldots, X_n]$.*

*Proof.* The existence of representation is proved by induction on essential arity. For essential arity 0, i.e. for constant functions, representation by constant polynomials is obvious. For a function $f : \{0,1\}^n \to B$ with essential arity $m > 0$, assuming the claim proved for lesser essential arities, and taking any index $i$ such that the $i$th

variable of $f$ is essential, let $f_0$ and $f_1$ be the $n$-ary $B$-valued functions given by

$$f_0(a_1,\ldots,a_n) = f(a_1,\ldots a_{i-1},0,a_{i+1},\ldots,a_n)$$
$$f_1(a_1,\ldots,a_n) = f(a_1,\ldots a_{i-1},1,a_{i+1},\ldots,a_n).$$

We have

$$f(a_1,\ldots,a_n) = (1-a_i^B)f_0(a_1,\ldots,a_n) + a_i^B f_1(a_1,\ldots,a_n)$$

and both $f_0$ and $f_1$ have essential arity less than $m$. By the induction hypothesis, $f_0$ and $f_1$ are represented by polynomials $p_0$ and $p_1$, respectively. Thus $f$ is represented by the polynomial

$$p = (1-X_i)p_0 + X_i p_1$$

and if $p$ had any powers of indeterminates $X_j^k$ with $k > 1$, by replacing each such occurrence by $X_j$ we would obtain a multilinear polynomial representing $f$.

Uniqueness is proved by contradiction. Suppose that $f$ had two distinct multilinear polynomial representations $p$ and $q$. Then the multilinear polynomial $p-q$ would represent the constant zero function. Let $J$ be a set of indices of smallest possible size, such that the monomial $c\prod_{j\in J} X_j$ occurs in $p-q$ with coefficient $c \neq 0_B$: such a $J$ must exist if $p-q$ is not the zero polynomial. But then the evaluation of $p-q$ at $(a_1,\ldots,a_n)$, where $a_j = 1_B$ if $j \in J$ and $a_j = 0_B$ otherwise, would be $c \neq 0_B$, contradicting the fact that $p-q$ represents the constant zero function. Thus $p-q$ must be the null polynomial, i.e. $p = q$. □

Let $f$ be a $B$-valued function on $\{0,1\}$, $f : \{0,1\}^n \to B$, where $B$ is a commutative ring with identity. The *degree* of $f$ is the smallest non-negative integer $d$ such that for every $J \subseteq \{1,\ldots,n\}$ of size $\mid J \mid > d$ the coefficient of $\prod_{j\in J} X_j$ in the multilinear polynomial representation of $f$ is zero. Thus the functions of degree 0 are precisely the constants (including the constant zero function).

**Theorem 6.** *If $B$ is any field of characteristic 2, and $k \geq 1$, then the class of $B$-valued functions on $\{0,1\}$ having degree less than $k$ is defined by the following functional equation (with vector variables $\mathbf{v}_1,\ldots,\mathbf{v}_k$):*

$$\sum_{I\subseteq\{1,\ldots,k\}} \mathbf{f}(\sum_{i\in I}\mathbf{v}_i) = 0 \qquad (3)$$

In (3) the inner summations refer to addition of vectors over the two-element field $\mathbf{GF}(2) = \{0,1\}$, while the outer summation refers to addition in the field $B$. For $I = \emptyset$, the empty sum $\sum_{i\in I}\mathbf{v}_i$ represents the constant zero.

*Proof.* First we prove that (3) is satisfied by every $B$-valued function on $\{0,1\}$ having degree less than $k$. From the form of the equation (3), it is easy to see that the class of functions satisfying (3) is closed under linear combinations with coefficients in $B$. Therefore, it is sufficient to prove that, for $n \geq 1$, every $n$-ary $B$-valued function $f$ on $\{0,1\}$ represented by a product of less than $k$ indeterminates, i.e. of the form $\prod_{j\in J} X_j$, $\mid J \mid < k$, $J \subseteq \{1,\ldots,n\}$, satisfies (3) .

Let $v_1, \ldots, v_k$ be any $n$-vectors in $\{0,1\}^n$. Let $w^J$ be the characteristic vector of $J$ in $\{0,1\}^n$, i.e. $w^J = (a_1, \ldots, a_n)$, where $a_j = 1$ if $j \in J$, and $a_j = 0$ otherwise. For every $I \subseteq \{1, \ldots, k\}$, consider the vector $w^J \cdot (\sum_{i \in I} v_i)$ in $\{0,1\}^n$, where the product $\cdot$ is defined componentwise. Observe that there are $2^k$ possible choices for $I$, yet due to the size of $\mid J \mid < k$, there at most $2^{k-1}$ distinct vectors of the form $w^J \cdot (\sum_{i \in I} v_i)$ in $\{0,1\}^n$. Therefore, there are distinct subsets $I_1$, $I_2$ of $\{1, \ldots, k\}$, such that

$$w^J \cdot (\sum_{i \in I_1} v_i) = w^J \cdot (\sum_{i \in I_2} v_i)$$

and for the symmetric difference $D$ of $I_1$ and $I_2$, we have

$$w^J \cdot (\sum_{i \in D} v_i) = 0$$

The $2^k$ subsets of $\{1, \ldots, k\}$, are matched into pairs $\{I, I + D\}$, where $I + D$ is the symmetric difference of $I$ and $D$, and because $f$ is represented by $\prod_{j \in J} X_j$, by the definition of $w^J$ it follows that for each such pair we have

$$f(\sum_{i \in I} v_i) = f(w^J \cdot (\sum_{i \in I} v_i)) = f(w^J \cdot (\sum_{i \in I+D} v_i)) = f(\sum_{i \in I+D} v_i)$$

Therefore, due to the fact that the underlying field $B$ has characteristic 2, the terms in the equation cancel pairwise.

Conversely, suppose now that the $n$-ary function $f$ is represented by a polynomial of degree greater than or equal to $k$. We show that $f$ does not satisfy the equation (3).

Let $g$ be the $B$-valued function on $\{0,1\}$ represented by the sum of those monomials in the polynomial representation of $f$ which have degree less than $k$. By the first part of the proof, $g$ satisfies (3). Working towards a contradiction, suppose that $f$ satisfies (3). Given the form of equation (3), this is the case if and only if the $n$-ary function $h = f + g$, represented by the sum of all monomials in the polynomial representation of $f$ having degree greater than or equal to $k$, satisfies (3).

Let $J$ be an inclusionwise minimal subset of $\{1, \ldots, n\}$, such that the monomial $c \prod_{j \in J} X_j$ appears in the polynomial representation of $h$ with coefficient $c \neq 0_B$. Note that $\mid J \mid \geq k$. We claim that if $f$ (or equivalently, $h$) satisfies (3), then the function $h_{\mathbf{k}}$ represented by the monomial $c \prod_{j \in \mathbf{k}} X_j$ where $\mathbf{k} = \{1, \ldots, k\}$, also satisfies equation (3).

Observe that, by the construction in the proof of Theorem 1, equation (3) is equivalent to a constraint $(R, S)$ whose antecedent $R$ is the range of a linear map with codomain $\mathbf{GF}(2)^m$, i.e. $R$ is a subspace of the vector space $\mathbf{GF}(2)^m$ over $\mathbf{GF}(2)$. Thus by Theorem 3 it follows that the class $\mathcal{K}$ of functions satisfying (3) is stable under right composition with the clone $\mathcal{L}_0$ of 0-preserving linear Boolean functions. In particular, $\mathcal{K}$ is closed under permutation and identification of variables, as well as under fixing variables to 0. It is not difficult to see that $h_{\mathbf{k}}$ can be

obtained from $h$ by a combination of these operations. In other words, if $h$ satisfies the equation (3), then $h_{\mathbf{k}}$ also satisfies the equation.

Now, let $v_1, \ldots, v_k$ be the unit $n$-vectors $e_1, \ldots, e_k$ in $\{0,1\}^n$. We have

$$\sum_{I \subseteq \mathbf{k}} h_{\mathbf{k}}(\sum_{i \in I} v_i) = h_{\mathbf{k}}(\sum_{i \in \mathbf{k}} v_i) = c \neq 0$$

which shows that $h_{\mathbf{k}}$ does not satisfy the equation (3), and yields the desired contradiction. □

In [1] it was shown that, for any positive integer $k$, the class of Boolean functions whose Zhegalkin polynomial has degree less than $k$, can be defined by "linear" equations. Theorem 6 above explicitly gives such an equation for every $k \geq 1$. For $k = 1$, the equation (3) can be rewritten as $\mathbf{f}(\mathbf{v}) = \mathbf{f}(\mathbf{0})$, and for $k = 2$, as $\mathbf{f}(\mathbf{v} + \mathbf{w}) = \mathbf{f}(\mathbf{v}) + \mathbf{f}(\mathbf{w}) + \mathbf{f}(\mathbf{0})$.

If $B$ is a field and $A = \{0,1\} = \mathbf{GF}(2)$, then a functional equation (1) is called *linear* if the functions $g_1, \ldots, g_m, g'_1, \ldots, g'_t$ are all affine maps from the $p$-dimensional vector space $\mathbf{GF}(2)^p$ to $\mathbf{GF}(2)$, and $h_1, h_2$ are affine maps from the $B$-vector spaces $B^m$ and $B^t$, respectively, to the scalar field $B$. (Recall that a function $F^n \to F$, where $F$ is any field, is affine if and only if it is of the form $(a_1, \ldots, a_n) \mapsto c_1 a_1 + \ldots c_n a_n + c$, for fixed scalars $c_1, \ldots, c_n, c$ in $F$.) Obviously, the functional equation (3) in Theorem 6 is linear. Our next result shows that the requirement on the characteristic of the underlying field is indeed essential.

**Theorem 7.** *For any field $B$ of characteristic different from 2, and any $k \geq 2$, the class of $B$-valued functions on $\{0,1\}$ having degree less than $k$ is not definable by any set of linear functional equations.*

*Proof.* As in the proof Theorem 6, if there would be a $k \geq 2$ such that the class $\mathcal{K}$ of $B$-valued functions on $\{0,1\}$ having degree less than $k$ is definable by some set of linear functional equations, then, using the construction given in the proof of Theorem 1, we would conclude that the class in question is definable by some set of constraints whose antecedents are affine subspaces of vector spaces over $\mathbf{GF}(2)$. These affine subspaces would be closed under the triple sum $u+v+w$, i.e. invariant under the clone $\mathcal{L}_{01}$ of constant-preserving linear Boolean functions. By Theorem 3, this would imply that $\mathcal{K}$ is stable under right composition with the clone $\mathcal{L}_{01}$. We show that this is not the case.

Consider the $(k-1)$-ary function $f$ represented by the monomial $X_1 \ldots X_{k-1}$. Let $\tau$ be the $(k+1)$-ary Boolean function in $\mathcal{L}_{01}$ given by

$$(a_1, \ldots, a_{k+1}) \mapsto a_{k-1} + a_k + a_{k+1}$$

Note that the $B$-valued function $\tau_B$ defined on $\{0,1\}$ which is valued $1_B$ on exactly those vectors $(a_1, \ldots, a_{k+1})$ for which $\tau(a_1, \ldots, a_{k+1}) = 1$ and valued $0_B$ otherwise, is represented by the polynomial

$$X_{k-1} + X_k + X_{k+1} - 2X_{k-1}X_k - 2X_kX_{k+1} - 2X_{k-1}X_{k+1} + 4X_{k-1}X_kX_{k+1}$$

where $+$ and $-$ are to be interpreted in $B$. Thus, the composition $f(f_1, \ldots, f_{k-1})$, where $f_{k-1} = \tau$ and $f_i$ is the $(k+1)$-ary $i$th projection function

$$(a_1, \ldots, a_{k+1}) \mapsto a_i$$

for $k = 1, \ldots, k-2$, is represented by the polynomial in $k+1$ indeterminates

$$X_1 \ldots X_{k-2}(X_{k-1} + X_k + X_{k+1} - \\ - 2X_{k-1}X_k - 2X_kX_{k+1} - 2X_{k-1}X_{k+1} + 4X_{k-1}X_kX_{k+1})$$

where $+$ and $-$ are to be interpreted in $B$. From the fact that $B$ has characteristic different from 2, it follows that this polynomial has degree greater than $k$. □

Note that for $k = 1$, the class of functions of degree less that $k$, i.e. the class of constants, is defined by the linear expression $\mathbf{f}(\mathbf{v}) = \mathbf{f}(0)$. In fact, from Theorem 7 above it follows that, if $B$ is any field of characteristic different from 2, then the set of constants is the only linearly definable class of $B$-valued functions on $\{0,1\}$ of bounded degree. However, Corollary 4 guarantees the existence of equational characterizations of these classes, because bounded degree classes are stable under right composition with the minimal clone $\mathcal{P}$ containing only projections. The following generalization of Corollary 3.3 in [4] provides an equation characterizing classes of bounded degree functions of Boolean variables, and whose codomain is any commutative ring with identity.

**Theorem 8.** *If $B$ is any commutative ring with identity, and $k \geq 1$, then the class of $B$-valued functions on $\{0,1\}$ having degree less than $k$ is defined by the following functional equation (with vector variables $\mathbf{v}_1, \ldots, \mathbf{v}_k$):*

$$\mathbf{f}(\bigwedge_{i \in \mathbf{k}} \mathbf{v}_i) + \sum_{\substack{I \subseteq \mathbf{k} \\ I \neq \emptyset}} (-1)^{|I|} \mathbf{f}(\bigvee_{j \in I} \bigwedge_{i \in \mathbf{k} \setminus \{j\}} \mathbf{v}_i) = 0 \qquad (4)$$

*where* $\mathbf{k} = \{1, \ldots, k\}$.

In (4) the summation refers to addition in the commutative ring $B$. Equation (4) was obtained in [4] as a combination of two opposite inequalities in the ordered real field $B = \mathbb{R}$. Inequalities are not available in general in a commutative ring, in particular in finite fields. However, the following direct proof, based on the principles used in establishing the functional inequality in Theorem 3.1 in [4], can still be used in the arbitrary commutative ring context.

*Proof.* First we show that every $B$-valued function on $\{0,1\}$ of degree less than $k$ satisfies equation (4). As in the proof of Theorem 6, it is enough to show that every monomial of degree less than $k$ satisfies equation (4), because every linear combination (with coefficients in $B$) of functions satisfying (4), also satisfies the equation.

Let $f$ be an $n$-ary $B$-valued function on $\{0,1\}$ represented by $\prod_{j \in J} X_j$, $| J | < k$, $J \subseteq \{1, \ldots, n\}$. Let $w^J$ be the characteristic vector of $J$ in $\{0,1\}^n$. Let $v_1, \ldots, v_k$

be any $n$-vectors in $\{0,1\}^n$, and let $u$ denote their conjunction $\bigwedge_{i\in\mathbf{k}} v_i$. For every $j \in \mathbf{k} = \{1,\ldots,k\}$, let

$$u_j = \bigwedge_{i\in\mathbf{k}\setminus\{j\}} v_i$$

and let the vector $z(I)$ be defined by

$$z(I) = w^J \cdot (\bigvee_{j\in I} u_j) \quad \text{for } \emptyset \neq I \subseteq \mathbf{k}, \quad \text{and} \quad z(\emptyset) = w^J \cdot u$$

where the product $\cdot$ is defined componentwise. From the fact that $k >\mid J \mid$, it follows that there is an $l \in \mathbf{k}$ such that

$$w^J \cdot u = w^J \cdot u_l$$

Fix such an index $l$. It is not difficult to see that, for every $I \subseteq \mathbf{k}$, we have

$$f(\bigvee_{j\in I} u_j) = f(z(I)) \quad \text{and} \quad z(I) = z(I + \{l\})$$

and thus the terms in the sum

$$f(\bigwedge_{i\in\mathbf{k}} v_i) + \sum_{\substack{I\subseteq\mathbf{k}\\ I\neq\emptyset}} (-1)^{|I|} f(\bigvee_{j\in I} u_j)$$

cancel pairwise, i.e. the sum is zero, which shows that $f$ satisfies (4).

In order to complete the proof of Theorem 8, we need to show that if $f$ is an $n$-ary function of degree greater than or equal to $k$, then equation (4) is not satisfied by $f$. Let $g$ and $h$ be the $n$-ary functions represented by the sum of monomials, in the polynomial representation of $f$, having degree less than $k$ and greater than or equal to $k$, respectively. As in the proof of Theorem 6, $f$ satisfies equation (4) if and only if $h$ satisfies the equation. We prove that $h$ does not satisfy (4).

Let $J$ be an inclusionwise minimal subset of $\mathbf{n} = \{1,\ldots,n\}$, such that the monomial $c\prod_{j\in J} X_j$ appears in the polynomial representation of $h$, with coefficient $c \neq 0_B$. Note that $\mid J \mid \geq k$. Let $J_0$ be any subset of $J$ of size $k$. For every $j \in J_0$, consider the $n$-vectors $y_j = (a_1,\ldots,a_n)$, where $a_j = 0$, $a_i = 0$ if $i \notin J$, and $a_i = 1$ if $i \in J \setminus \{j\}$. Let $v_1,\ldots,v_k$ be defined as the vectors $y_j$, $j \in J_0$, in any order. Let $u = \bigwedge_{i\in\mathbf{k}} v_i$, and for each $j \in \mathbf{k}$, let

$$u_j = \bigwedge_{i\in\mathbf{k}\setminus\{j\}} v_i$$

Observe that for $I \subseteq \mathbf{k}$, all monomials in the polynomial representation of $h$ are evaluated to zero on

$$\bigvee_{j\in I} u_j$$

except in the case $I = \mathbf{k}$, where the only monomial which has non-zero value is $c\prod_{j\in J} X_j$, because the $n$-vector

$$\bigvee_{j\in\mathbf{k}} u_j = (a_1, \ldots, a_n)$$

is given by $a_t = 1$ if $t \in J$, and $a_t = 0$ otherwise. Therefore, we have

$$h(\bigwedge_{i\in\mathbf{k}} v_i) + \sum_{\substack{I\subseteq\mathbf{k}\\ I\neq\emptyset}} (-1)^{|I|} h(\bigvee_{j\in I} u_j) = (-1)^k h(\bigvee_{j\in\mathbf{k}} u_j) = (-1)^k c \neq 0$$

which shows that $h$, and thus $f$, does not satisfy equation (4). □

Theorem 8 provides in particular an alternative equational characterization of classes of Boolean functions whose Zhegalkin polynomials have degree bounded by a positive integer $k$.

# References

[1] M. Couceiro, S. Foldes. "Definability of Boolean Function Classes by Linear Equations over GF(2)", *Discrete Applied Mathematics* 142 (2004) 29–34.

[2] M. Couceiro, S. Foldes. "On Closed Sets of Relational Constraints and Classes of Functions Closed under Variable Substitutions", *Algebra Universalis*, **54** (2005) 149–165.

[3] O. Ekin, S. Foldes, P.L. Hammer, L. Hellerstein. "Equational Characterizations of Boolean Functions Classes", *Discrete Mathematics* 211 (2000) 27–51.

[4] S. Foldes, P.L. Hammer. "Submodularity, Supermodularity and Higher Order Monotonicities of Pseudo-Boolean Functions", *Mathematics of Operations Research* **30** 2 (2005) 453-461.

[5] S. Foldes, G. R. Pogosyan. "Post classes characterized by functional terms", *Discrete Applied Mathematics* 142 (2004) 3551.

[6] R. Godement. *Algebra*, Kershaw Publishing Company, 1969.

[7] P.L. Hammer. S. Rudeanu. *Boolean Methods in Operations Research and Related Areas*, Springer 1968.

[8] N. Pippenger. "Galois Theory for Minors of Finite Functions", *Discrete Mathematics* 254 (2002) 405–419.

[9] G. R. Pogosyan. "Classes of Boolean Functions Defined by Functional Terms", *Multiple - Valued Logic* **7** 5–6 (2001) 417–448.

[10] R. Pöschel. "Concrete Representation of Algebraic Structures and a General Galois Theory", *Contributions to General Algebra*, Proceedings Klagenfurt Conference, May 25-28 (1978) 249–272. Verlag J. Heyn, Klagenfurt, Austria 1979.

[11] R. Pöschel. "A General Galois Theory for Operations and Relations and Concrete Characterization of Related Algebraic Structures", Report R-01/80. *Zentralinstitut für Math. und Mech.*, Berlin 1980.

[12] C. Wang. "Boolean Minors", *Discrete Mathematics* 141 (1995) 237–258.

[13] C. Wang, A.C. Williams. "The Threshold Order of a Boolean Function", *Discrete Applied Mathematics* 31 (1991) 51–69.

[14] D. J. A. Welsh. *Matroid Theory*, Academic Press, 1976.

[15] I. E. Zverovich. "Characterization of Closed Classes of Boolean Functions in Terms of Forbidden Subfunctions and Post Classes", *Discrete Applied Mathematics* 149 (2005) 200-218.

Acta Cybernetica 18 (2007) 77–113.

# Intuitionistic computability logic*

Giorgi Japaridze†

**Abstract**

Computability logic (CL) is a systematic formal theory of computational tasks and resources, which, in a sense, can be seen as a semantics-based alternative to (the syntactically introduced) linear logic. With its expressive and flexible language, where formulas represent computational problems and "truth" is understood as algorithmic solvability, CL potentially offers a comprehensive logical basis for constructive applied theories and computing systems inherently requiring constructive and computationally meaningful underlying logics. Among the best known constructivistic logics is Heyting's intuitionistic calculus **INT**, whose language can be seen as a special fragment of that of CL. The constructivistic philosophy of **INT**, however, just like the resource philosophy of linear logic, has never really found an intuitively convincing and mathematically strict semantical justification. CL has good claims to provide such a justification and hence a materialization of Kolmogorov's known thesis "**INT** = logic of problems". The present paper contains a soundness proof for **INT** with respect to the CL semantics.

**Keywords:** computability logic, interactive computation, game semantics, linear logic, intuitionistic logic

## 1 Introduction

*Computability logic* (CL), introduced recently in [7], is a formal theory of computability in the same sense as classical logic is a formal theory of truth. It understands formulas as (interactive) computational problems, and their "truth" as algorithmic solvability. Computational problems, in turn, are defined as games played by a machine against the environment, with algorithmic solvability meaning existence of a machine that always wins the game.

Intuitionistic computability logic is not a modification or version of CL. The latter takes pride in its universal applicability, stability and "immunity to possible future revisions and tampering" ([7], p. 12). Rather, what we refer to as *intuitionistic computability logic* is just a — relatively modest — fragment of CL, obtained

*This material is based upon work supported by the National Science Foundation under Grant No. 0208816

†Computing Sciences Department, Villanova University, 800 Lancaster Avenue, Villanova, PA 19085, USA E-mail: giorgi.japaridze@villanova.edu

by mechanically restricting its formalism to a special sublanguage. It was conjectured in [7] that the (set of the valid formulas of the) resulting fragment of CL is described by Heyting's *intuitionistic calculus* **INT**. The present paper is devoted to a verification of the soundness part of that conjecture.

Bringing **INT** and CL together could signify a step forward not only in logic but also in theoretical computer science. **INT** has been attracting the attention of computer scientists since long ago. And not only due to the beautiful phenomenon within the 'formulas-as-types' approach known as the Curry-Howard isomorphism. **INT** appears to be an appealing alternative to classical logic within the more traditional approaches as well. This is due to the general constructive features of its deductive machinery, and Kolmogorov's [14] well-known yet so far rather abstract thesis according to which intuitionistic logic is (or should be) a logic of problems. The latter inspired many attempts to find a "problem semantics" for the language of intuitionistic logic [5, 13, 16], none of which, however, has fully succeeded in justifying **INT** as a logic of problems. Finding a semantical justification for **INT** was also among the main motivations for Lorenzen [15], who pioneered game-semantical approaches in logic. After a couple of decades of trial and error, the goal of obtaining soundness and completeness of **INT** with respect to Lorenzen's game semantics was achieved [3]. The value of such an achievement is, however, dubious, as it came as a result of carefully tuning the semantics and adjusting it to the goal at the cost of sacrificing some natural intuitions that a game semantics could potentially offer.[1] After all, some sort of a specially designed technical semantics can be found for virtually every formal system, but the whole question is how natural and usable such a semantics is in its own right. In contrast, the CL semantics was elaborated without any target deductive construction in mind, following the motto "*Axiomatizations should serve meaningful semantics rather than vice versa*". Only retroactively was it observed that the semantics of CL yields logics similar to or identical with some known axiomatically introduced constructivistic logics such as linear logic or **INT**. Discussions given in [7, 8, 10, 11] demonstrate how naturally the semantics of CL emerges and how much utility it offers, with potential application areas ranging from the pure theory of (interactive) computation to knowledgebase systems, systems for planning and action, and constructive applied theories. As this semantics has well-justified claims to be a semantics of computational problems, the results of the present article speak strongly in favor of Kolmogorov's thesis, with a promise of a full materialization of the thesis in case a completeness proof of **INT** is also found.

The main utility of the present result is in the possibility to base applied theories or knowledgebase systems on **INT**. Nonlogical axioms — or the knowledge base — of such a system would be any collection of (formulas expressing) problems whose

[1] Using Blass's [2] words, 'Supplementary rules governing repeated attacks and defenses were devised by Lorenzen so that the formulas for which $P$ [proponent] has a winning strategy are exactly the intuitionistically provable ones'. Quoting [6], 'Lorenzen's approach describes logical validity exclusively in terms of rules without appealing to any kind of truth values for atoms, and this makes the semantics somewhat vicious ... as it looks like just a "pure" syntax rather than a semantics'.

algorithmic solutions are known. Then, our soundness theorem for **INT** — which comes in a strong form called uniform-constructive soundness — guarantees that every theorem $T$ of the theory also has an algorithmic solution and, furthermore, such a solution can be effectively constructed from a proof of $T$. This makes **INT** a problem-solving tool: finding a solution for a given problem reduces to finding a proof of that problem in the theory.

It is not an ambition of the present paper to motivationally (re)introduce and (re)justify computability logic and its intuitionistic fragment in particular. This job has been done in [7] and once again — in a more compact way — in [10]. An assumption is that the reader is familiar with at least the motivational/philosophical parts of either paper and this is why (s)he decided to read the present article. While helpful in fully understanding the import of the present results, from the purely technical point of view such a familiarity, however, is not necessary, as this paper provides all necessary definitions. Even if so, [7] and/or [10] could still help a less advanced reader in getting a better hold of the basic technical concepts. Those papers are written in a semitutorial style, containing ample examples, explanations and illustrations, with [10] even including exercises.

# 2 A brief informal overview of some basic concepts

As noted, formulas of CL represent interactive computational problems. Such problems are understood as games between two players: $\top$, called **machine**, and $\bot$, called **environment**. $\top$ is a mechanical device with a fully determined, algorithmic behavior. On the other hand, there are no restrictions on the behavior of $\bot$. A problem/game is considered (algorithmically) solvable/winnable iff there is a machine that wins the game no matter how the environment acts.

Logical operators are understood as operations on games/problems. One of the important groups of such operations, called **choice operations**, consists of $\sqcap, \sqcup, \bigsqcap, \bigsqcup$, in our present approach corresponding to the intuitionistic operators of conjunction, disjunction, universal quantifier and existential quantifier, respectively. $A_1 \sqcap \ldots \sqcap A_n$ is a game where the first legal move ("choice"), which should be one of the elements of $\{1, \ldots, n\}$, is by the environment. After such a move/choice $i$ is made, the play continues and the winner is determined according to the rules of $A_i$; if a choice is never made, $\bot$ loses. $A_1 \sqcup \ldots \sqcup A_n$ is defined in a symmetric way with the roles of $\bot$ and $\top$ interchanged: here it is $\top$ who makes an initial choice and who loses if such a choice is not made. With the universe of discourse being $\{1, 2, 3, \ldots\}$, the meanings of the "big brothers" $\bigsqcap$ and $\bigsqcup$ of $\sqcap$ and $\sqcup$ can now be explained by $\bigsqcap x A(x) = A(1) \sqcap A(2) \sqcap A(3) \sqcap \ldots$ and $\bigsqcup x A(x) = A(1) \sqcup A(2) \sqcup A(3) \sqcup \ldots$.

The remaining two operators of intuitionistic logic are the binary $\circ\!\!-$ ("intuitionistic implication") and the 0-ary \$ ("intuitionistic absurd"), with the intuitionistic negation of $F$ simply understood as an abbreviation for $F \circ\!\!- \$$. The intuitive meanings of $\circ\!\!-$ and \$ are "reduction" (in the weakest possible sense) and "a problem of universal strength", respectively. In what precise sense is \$ a universal-strength problem will be seen in Section 6. As for $\circ\!\!-$, its meaning can

be better explained in terms of some other, more basic, operations of CL that have no official intuitionistic counterparts.

One group of such operations comprises **negation** $\neg$ and the so called **parallel operations** $\wedge, \vee, \rightarrow$. Applying $\neg$ to a game $A$ interchanges the roles of the two players: $\top$'s moves and wins become $\bot$'s moves and wins, and vice versa. Say, if *Chess* is the game of chess from the point of view of the white player, then $\neg$*Chess* is the same game as seen by the black player. Playing $A_1 \wedge \ldots \wedge A_n$ (resp. $A_1 \vee \ldots \vee A_n$) means playing the $n$ games in parallel where, in order to win, $\top$ needs to win in all (resp. at least one) of the components $A_i$. Back to our chess example, the two-board game *Chess* $\vee \neg$*Chess* can be easily won by just mimicking in *Chess* the moves made by the adversary in $\neg$*Chess* and vice versa. On the other hand, winning *Chess*$\sqcup\neg$*Chess* is not easy at all: here $\top$ needs to choose between *Chess* and $\neg$*Chess* (i.e. between playing white or black), and then win the chosen one-board game. Technically, a move $\alpha$ in the $k$th $\wedge$-conjunct or $\vee$-disjunct is made by prefixing $\alpha$ with '$k$.'. For example, in (the initial position of) $(A \sqcup B) \vee (C \sqcap D)$, the move '2.1' is legal for $\bot$, meaning choosing the first $\sqcap$-conjunct in the second $\vee$-disjunct of the game. If such a move is made, the game will continue as $(A \sqcup B) \vee C$. One of the distinguishing features of CL games from the more traditional concepts of games ([1, 2, 3, 6, 15]) is the absence of *procedural rules* — rules strictly regulating which of the players can or should move in any given situation. E.g., in the above game $(A \sqcup B) \vee (C \sqcap D)$, $\top$ also has legal moves — the moves '1.1' and '1.2'. In such cases CL allows either player to move, depending on who wants or can act faster.[2] As argued in [7] (Section 3), only this "free" approach makes it possible to adequately capture certain natural intuitions such as truly parallel/concurrent computations.

The operation $\rightarrow$ is defined by $A \rightarrow B = (\neg A) \vee B$. Intuitively, this is the problem of *reducing* $B$ to $A$: solving $A \rightarrow B$ means solving $B$ having $A$ as an external *computational resource*. Resources are symmetric to problems: what is a problem to solve for one player is a resource that the other player can use, and vice versa. Since $A$ is negated in $(\neg A) \vee B$ and negation means switching the roles, $A$ appears as a resource rather than problem for $\top$ in $A \rightarrow B$. To get a feel of $\rightarrow$ as a problem reduction operation, the following — already "classical" in CL — example may help. Let, for any $m, n$, *Accepts*$(m, n)$ mean the game where none of the players has legal moves, and which is automatically won by $\top$ if Turing machine $m$ accepts input $n$, and otherwise automatically lost. This sort of zero-length games are called **elementary** in CL, which understands every classical proposition/predicate as an elementary game and vice versa, with "true"="won by $\top$" and "false"="lost by $\top$". Note that then $\sqcap x \sqcap y\big(\mathit{Accepts}(x, y) \sqcup \neg \mathit{Accepts}(x, y)\big)$ expresses the acceptance problem as a decision problem: in order to win, the machine should be able to tell whether $x$ accepts $y$ or not (i.e., choose the true disjunct) for any particular values for $x$ and $y$ selected by the environment. This problem is undecidable, which obviously means that there is no machine that (always) wins

[2] This is true for the case when the underlying model of computation is HPM (see Section 5), but seemingly not so when it is EPM — the model employed in the present paper. It should be remembered, however, that EPM is viewed as a secondary model in CL, admitted only due to the fact that it has been proven ([7]) to be equivalent to the basic HPM model.

the game $\sqcap x \sqcap y\big(Accepts(x,y) \sqcup \neg Accepts(x,y)\big)$. However, the acceptance problem is known to be algorithmically reducible to the halting problem. The latter can be expressed by $\sqcap x \sqcap y\big(Halts(x,y) \sqcup \neg Halts(x,y)\big)$, with the obvious meaning of the elementary game/predicate $Halts(x,y)$. This reducibility translates into our terms as existence of a machine that wins

$$\sqcap x \sqcap y\big(Halts(x,y) \sqcup \neg Halts(x,y)\big) \rightarrow \sqcap x \sqcap y\big(Accepts(x,y) \sqcup \neg Accepts(x,y)\big). \quad (1)$$

Such a machine indeed exists. A successful strategy for it is as follows. At the beginning, $\top$ waits till $\bot$ specifies some values $m$ and $n$ for $x$ and $y$ in the consequent, i.e. makes the moves '$2.m$' and '$2.n$'. Such moves, bringing the consequent down to $Accepts(m,n) \sqcup \neg Accepts(m,n)$, can be seen as asking the question "does machine $m$ accept input $n$?". To this question $\top$ replies by the counterquestion "does $m$ halt on $n$?", i.e. makes the moves '$1.m$ and '$1.n$', bringing the antecedent down to $Halts(m,n) \sqcup \neg Halts(m,n)$. The environment has to correctly answer this counterquestion, or else it loses. If it answers "no" (i.e. makes the move '1.2' and thus further brings the antecedent down to $\neg Halts(m,n)$), $\top$ also answers "no" to the original question in the consequent (i.e. makes the move '2.2'), with the overall game having evolved to the true and hence $\top$-won proposition/elementary game $\neg Halts(m,n) \rightarrow \neg Accepts(m,n)$. Otherwise, if the environment's answer is "yes" (move '1.1'), $\top$ simulates Turing machine $m$ on input $n$ until it halts, and then makes the move '2.1' or '2.2' depending whether the simulation accepted or rejected.

Various sorts of reduction have been defined and studied in an ad hoc manner in the literature. A strong case can be made in favor of the thesis that the reduction captured by our $\rightarrow$ is the most basic one, with all other reasonable concepts of reduction being definable in terms of $\rightarrow$. Most natural of those concepts is the one captured by the earlier-mentioned operation of "intuitionistic implication" $\circ\!\!-$, with $A \circ\!\!- B$ defined in terms of $\rightarrow$ and (yet another natural operation) $\downarrow\!\!\circ$ by $A \circ\!\!- B = (\downarrow\!\!\circ A) \rightarrow B$. What makes $\circ\!\!-$ so natural is that it captures our intuition of reducing one problem to another in the weakest possible sense. The well-established concept of Turing reduction has the same claim. But the latter is only defined for non-interactive, two-step (question/answer, or input/output) problems, such as the above halting or acceptance problems. When restricted to this sort of problems, as one might expect, $\circ\!\!-$ indeed turns out to be equivalent to Turing reduction. The former, however, is more general than the latter as it is applicable to all problems regardless their forms and degrees of interactivity. Turing reducibility of a problem $B$ to a problem $A$ is defined as the possibility to algorithmically solve $B$ having an oracle for $A$. Back to (1), the role of $\bot$ in the antecedent is in fact that of an oracle for the halting problem. Notice, however, that the usage of the oracle is limited there as it only can be employed once: after querying regarding whether $m$ halts of $n$, the machine would not be able to repeat the same query with different parameters $m'$ and $n'$, for that would require two "copies" of $\sqcap x \sqcap y\big(Halts(x,y) \sqcup \neg Halts(x,y)\big)$ rather than one. On the other hand, Turing reduction to $A$ and, similarly, our $A \circ\!\!- \ldots$, allow unlimited and recurring usage of $A$, which the resource-conscious CL understands as $\rightarrow$-reduction not to $A$

but to the stronger problem expressed by $\S A$, called the **branching recurrence** of $A$.[3] Two more recurrence operations have been introduced within the framework of CL ([10]): *parallel recurrence* $\curlywedge$ and *sequential recurrence* $\barwedge$. Common to all of these operations is that, when applied to a resource $A$, they turn it into a resource that allows to reuse $A$ an unbounded number of times. The difference is in how "reusage" is exactly understood. Imagine a computer that has a program successfully playing *Chess*. The resource that such a computer provides is obviously something stronger than just *Chess*, for it allows to play *Chess* as many times as the user wishes, while *Chess*, as such, only assumes one play. The simplest operating system would allow to start a session of *Chess*, then — after finishing or abandoning and destroying it — start a new play again, and so on. The game that such a system plays — i.e. the resource that it supports/provides — is $\barwedge$*Chess*, which assumes an unbounded number of plays of *Chess* in a sequential fashion. However, a more advanced operating system would not require to destroy the old session(s) before starting a new one; rather, it would allow to run as many parallel sessions as the user needs. This is what is captured by $\curlywedge$*Chess*, meaning nothing but the infinite conjunction *Chess* $\wedge$ *Chess* $\wedge \ldots$. As a resource, $\curlywedge$*Chess* is obviously stronger than $\barwedge$*Chess* as it gives the user more flexibility. But $\curlywedge$ is still not the strongest form of reusage. A really good operating system would not only allow the user to start new sessions of *Chess* without destroying old ones; it would also make it possible to branch/replicate each particular session, i.e. create any number of "copies" of any already reached position of the multiple parallel plays of *Chess*, thus giving the user the possibility to try different continuations from the same position. After analyzing the formal definition of $\S$ given in Section 3 — or, better, the explanations provided in Section 13 of [7] — the reader will see that $\S$*Chess* is exactly what accounts for this sort of a situation. $\curlywedge$*Chess* can then be thought of as a restricted version of $\S$*Chess* where only the initial position can be replicated. A well-justified claim can be made that $\S A$ captures our strongest possible intuition of "recycling"/"reusing" $A$. This automatically translates into another claim, according to which $A \circ\!\!- B$, i.e. $\S A \rightarrow B$, captures our weakest possible — and hence most natural — intuition of reducing $B$ to $A$.

As one may expect, the three concepts of recurrence validate different principles. For example, one can show that the left $\sqcup$- or $\sqcup$-introduction rules of **INT**, which are sound with $A \circ\!\!- B$ understood as $\S A \rightarrow B$, would fail if $A \circ\!\!- B$ was understood as $\curlywedge A \rightarrow B$ or $\barwedge A \rightarrow B$. A naive person familiar with linear logic and seeing philosophy-level connections between our recurrence operations and Girard's [4] *storage* operator !, might ask which of the three recurrence operations "corresponds" to !. In the absence of a clear resource semantics for linear logic, perhaps such a question would not be quite meaningful though. Closest to our present approach is that of [1], where Blass proved soundness for the propositional fragment of **INT** with respect to his semantics, reintroduced 20 years later [2] in the new context of linear logic.

[3]The term "branching recurrence" and the symbols $\S$ and $\circ\!\!-$ were established in [10]. The earlier paper [7] uses "branching conjunction", ! and $\Rightarrow$ instead. In the present paper, $\Rightarrow$ has a different meaning — that of a separator of the two parts of a sequent.

To appreciate the difference between $\rightarrow$ and $\circ\!\!-$, let us remember the Kolmogorov complexity problem. It can be expressed by $\sqcap u \sqcup z K(z,u)$, where $K(z,u)$ is the predicate "$z$ is the size of the smallest (code of a) Turing machine that returns $u$ on input 1". Just like the acceptance problem, the Kolmogorov complexity problem has no algorithmic solution but is algorithmically reducible to the halting problem. However, such a reduction can be shown to essentially require recurring usage of the resource $\sqcap x \sqcap y\big(\mathit{Halts}(x,y) \sqcup \neg \mathit{Halts}(x,y)\big)$. That is, while the following game is winnable by a machine, it is not so with $\rightarrow$ instead of $\circ\!\!-$:

$$\sqcap x \sqcap y\big(\mathit{Halts}(x,y) \sqcup \neg \mathit{Halts}(x,y)\big) \circ\!\!- \sqcap u \sqcup z K(z,u). \tag{2}$$

Here is $\top$'s strategy for (2) in relaxed terms: $\top$ waits till $\bot$ selects a value $m$ for $u$ in the consequent, thus asking $\top$ the question "what is the Kolmogorov complexity of $m$?". After this, starting from $i = 1$, $\top$ does the following: it creates a new copy of the (original) antecedent, and makes the two moves in it specifying $x$ and $y$ as $i$ and 1, respectively, thus asking the counterquestion "does machine $i$ halt on input 1?". If $\bot$ responds by choosing $\neg\mathit{Halts}(i,1)$ ("no"), $\top$ increments $i$ by one and repeats the step; otherwise, if $\bot$ responds by $\mathit{Halts}(i,1)$ ("yes"), $\top$ simulates machine $i$ on input 1 until it halts; if it sees that machine $i$ returned $m$, it makes the move in the consequent specifying $z$ as $|i|$ (here $|i|$ means the size of $i$, i.e., $|i| = \log_2 i$), thus saying that $|i|$ is the Kolmogorov complexity of $m$; otherwise, it increments $i$ by one and repeats the step.

# 3 Constant games

Now we are getting down to formal definitions of the concepts informally explained in the previous section. Our ultimate concept of games will be defined in the next section in terms of the simpler and more basic class of games called constant games. To define this class, we need some technical terms and conventions. Let us agree that by a **move** we mean any finite string over the standard keyboard alphabet. One of the non-numeric and non-punctuation symbols of the alphabet, denoted ♠, is designated as a special-status move, intuitively meaning a move that is always illegal to make. A **labeled move** (**labmove**) is a move prefixed with $\top$ or $\bot$, with its prefix (**label**) indicating which player has made the move. A **run** is a (finite or infinite) sequence of labeled moves, and a **position** is a finite run.

**Convention 1.** We will be exclusively using the letters $\Gamma, \Theta, \Phi, \Psi, \Upsilon$ for runs, $\wp$ for players, $\alpha, \beta, \gamma$ for moves, and $\lambda$ for labmoves. Runs will be often delimited with "$\langle$" and "$\rangle$", with $\langle\rangle$ thus denoting the **empty run**. The meaning of an expression such as $\langle \Phi, \wp\alpha, \Gamma\rangle$ must be clear: this is the result of appending to position $\langle\Phi\rangle$ the labmove $\langle\wp\alpha\rangle$ and then the run $\langle\Gamma\rangle$. $\neg\Gamma$ (not to confuse this $\neg$ with the same-shape game operation of negation) will mean the result of simultaneously replacing every label $\top$ in every labmove of $\Gamma$ by $\bot$ and vice versa. Another important notational convention is that, for a string/move $\alpha$, $\Gamma^{\alpha}$ means the result of removing from $\Gamma$ all labmoves except those of the form $\wp\alpha\beta$, and then deleting the prefix '$\alpha$' in the remaining moves, i.e. replacing each such $\wp\alpha\beta$ by $\wp\beta$.

The following item is a formal definition of constant games combined with some less formal conventions regarding the usage of certain terminology.

**Definition 2.** *A* **constant game** *is a pair* $A = (\mathbf{Lr}^A, \mathbf{Wn}^A)$, *where:*

1. $\mathbf{Lr}^A$ *is a set of runs not containing (whatever-labeled) move* ♠, *satisfying the condition that a (finite or infinite) run is in* $\mathbf{Lr}^A$ *iff all of its nonempty finite — not necessarily proper — initial segments are in* $\mathbf{Lr}^A$ *(notice that this implies* $\langle\rangle \in \mathbf{Lr}^A$*). The elements of* $\mathbf{Lr}^A$ *are said to be* **legal runs** *of* $A$*, and all other runs are said to be* **illegal**. *We say that* $\alpha$ *is a* **legal move** *for* $\wp$ *in a position* $\Phi$ *of* $A$ *iff* $\langle \Phi, \wp\alpha\rangle \in \mathbf{Lr}^A$*; otherwise* $\alpha$ *is* **illegal**. *When the last move of the shortest illegal initial segment of* $\Gamma$ *is* $\wp$*-labeled, we say that* $\Gamma$ *is a* $\wp$**-illegal** *run of* $A$.
2. $\mathbf{Wn}^A$ *is a function that sends every run* $\Gamma$ *to one of the players* $\top$ *or* $\bot$, *satisfying the condition that if* $\Gamma$ *is a* $\wp$*-illegal run of* $A$, *then* $\mathbf{Wn}^A\langle\Gamma\rangle \neq \wp$. *When* $\mathbf{Wn}^A\langle\Gamma\rangle = \wp$, *we say that* $\Gamma$ *is a* $\wp$**-won** *(or* **won by** $\wp$*) run of* $A$*; otherwise* $\Gamma$ *is* **lost** *by* $\wp$. *Thus, an illegal run is always lost by the player who has made the first illegal move in it.*

**Definition 3.** *Let* $A$, $B$, $A_1, A_2, \ldots$ *be constant games, and* $n \in \{2, 3, \ldots\}$.

1. $\neg A$ *is defined by:* $\Gamma \in \mathbf{Lr}^{\neg A}$ *iff* $\neg\Gamma \in \mathbf{Lr}^A$; $\mathbf{Wn}^{\neg A}\langle\Gamma\rangle = \top$ *iff* $\mathbf{Wn}^A\langle\neg\Gamma\rangle = \bot$.
2. $A_1 \sqcap \ldots \sqcap A_n$ *is defined by:* $\Gamma \in \mathbf{Lr}^{A_1\sqcap\ldots\sqcap A_n}$ *iff* $\Gamma = \langle\rangle$ *or* $\Gamma = \langle \bot i, \Theta\rangle$, *where* $i \in \{1, \ldots, n\}$ *and* $\Theta \in \mathbf{Lr}^{A_i}$; $\mathbf{Wn}^{A_1\sqcap\ldots\sqcap A_n}\langle\Gamma\rangle = \bot$ *iff* $\Gamma = \langle \bot i, \Theta\rangle$, *where* $i \in \{1, \ldots, n\}$ *and* $\mathbf{Wn}^{A_i}\langle\Theta\rangle = \bot$.
3. $A_1 \wedge \ldots \wedge A_n$ *is defined by:* $\Gamma \in \mathbf{Lr}^{A_1\wedge\ldots\wedge A_n}$ *iff every move of* $\Gamma$ *starts with 'i.' for one of the* $i \in \{1, \ldots, n\}$ *and, for each such* $i$, $\Gamma^{i.} \in \mathbf{Lr}^{A_i}$; *whenever* $\Gamma \in \mathbf{Lr}^{A_1\wedge\ldots\wedge A_n}$, $\mathbf{Wn}^{A_1\wedge\ldots\wedge A_n}\langle\Gamma\rangle = \top$ *iff, for each* $i \in \{1, \ldots, n\}$, $\mathbf{Wn}^{A_i}\langle\Gamma^{i.}\rangle = \top$.
4. $A_1 \sqcup \ldots \sqcup A_n$ *and* $A_1 \vee \ldots \vee A_n$ *are defined exactly as* $A_1 \sqcap \ldots \sqcap A_n$ *and* $A_1 \wedge \ldots \wedge A_n$, *respectively, only with "*$\top$*" and "*$\bot$*" interchanged. And* $A \rightarrow B$ *is defined as* $(\neg A) \vee B$.
5. *The infinite* $\sqcap$*-conjunction* $A_1 \sqcap A_2 \sqcap \ldots$ *is defined exactly as* $A_1 \sqcap \ldots \sqcap A_n$, *only with "*$i \in \{1, 2, \ldots\}$*" instead of "*$i \in \{1, \ldots, n\}$*". Similarly for the infinite versions of* $\sqcup$, $\wedge$, $\vee$.
6. *In addition to the earlier-established meanings, the symbols* $\top$ *and* $\bot$ *also denote two special — simplest — games, defined by* $\mathbf{Lr}^\top = \mathbf{Lr}^\bot = \{\langle\rangle\}$, $\mathbf{Wn}^\top\langle\rangle = \top$ *and* $\mathbf{Wn}^\bot\langle\rangle = \bot$.

An important operation not explicitly mentioned in Section 2 is what is called *prefixation*. This operation takes two arguments: a constant game $A$ and a position $\Phi$ that must be a legal position of $A$ (otherwise the operation is undefined), and returns the game $\langle\Phi\rangle A$. Intuitively, $\langle\Phi\rangle A$ is the game playing which means playing $A$ starting (continuing) from position $\Phi$. That is, $\langle\Phi\rangle A$ is the game to which $A$ **evolves** (will be "**brought down**") after the moves of $\Phi$ have been made. We have already used this intuition when explaining the meaning of choice operations in Section 2: we said that after $\bot$ makes an initial move $i \in \{1, \ldots, n\}$, the game

$A_1 \sqcap \ldots \sqcap A_n$ continues as $A_i$. What this meant was nothing but that $(\bot i)(A_1 \sqcap \ldots \sqcap A_n) = A_i$. Similarly, $(\top i)(A_1 \sqcup \ldots \sqcup A_n) = A_i$. Here is the definition of prefixation:

**Definition 4.** *Let $A$ be a constant game and $\Phi$ a legal position of $A$. The game $\langle\Phi\rangle A$ is defined by:* $\mathbf{Lr}^{\langle\Phi\rangle A} = \{\Gamma \mid \langle\Phi,\Gamma\rangle \in \mathbf{Lr}^A\}$; $\mathbf{Wn}^{\langle\Phi\rangle A}\langle\Gamma\rangle = \mathbf{Wn}^A\langle\Phi,\Gamma\rangle$.

The operation $\mathbin{\downarrow\!\!\circ}$ is somewhat more complex and its definition relies on certain additional conventions. We will be referring to (possibly infinite) strings of 0s and 1s as **bit strings**, using the letters $w$, $u$ as metavariables for them. The expression $wu$, meaningful only when $w$ is finite, will stand for the concatenation of strings $w$ and $u$. We write $w \preceq u$ to mean that $w$ is a (not necessarily proper) initial segment of $u$. The letter $\epsilon$ will exclusively stand for the **empty bit string**.

**Convention 5.** By a **tree** we mean a nonempty set $T$ of bit strings, called **branches** of the tree, such that, for every $w, u$, we have: (a) if $w \in T$ and $u \preceq w$, then $u \in T$; (b) $w0 \in T$ iff $w1 \in T$; (c) if $w$ is infinite and every finite $u$ with $u \preceq w$ is in $T$, then $w \in T$. Note that $T$ is finite iff every branch of it is finite. A **complete branch** of $T$ is a branch $w$ such that no bit string $u$ with $w \preceq u \neq w$ is in $T$. Finite branches are called **nodes**, and complete finite branches called **leaves**.

**Definition 6.** *We define the notion of a* **prelegal position**, *together with the function* Tree *that associates a finite tree* $\mathrm{Tree}\langle\Phi\rangle$ *with each prelegal position $\Phi$, by the following induction:*

1. $\langle\rangle$ *is a prelegal position, and* $\mathrm{Tree}\langle\rangle = \{\epsilon\}$.
2. $\langle\Phi,\lambda\rangle$ *is a prelegal position iff $\Phi$ is so and one of the following two conditions is satisfied:*
   a) $\lambda = \bot w$*: for some leaf $w$ of* $\mathrm{Tree}\langle\Phi\rangle$. *We call this sort of a labmove $\lambda$* **replicative**. *In this case* $\mathrm{Tree}\langle\Phi,\lambda\rangle = \mathrm{Tree}\langle\Phi\rangle \cup \{w0, w1\}$.
   b) *$\lambda$ is $\bot w.\alpha$ or $\top w.\alpha$ for some node $w$ of* $\mathrm{Tree}\langle\Phi\rangle$ *and move $\alpha$. We call this sort of a labmove $\lambda$* **nonreplicative**. *In this case* $\mathrm{Tree}\langle\Phi,\lambda\rangle = \mathrm{Tree}\langle\Phi\rangle$.

*The terms "replicative" and "nonreplicative" also extend from labmoves to moves. When a run $\Gamma$ is infinite, it is considered prelegal iff all of its finite initial segments are so. For such a $\Gamma$, the value of* $\mathrm{Tree}\langle\Gamma\rangle$ *is the smallest tree such that, for every finite initial segment $\Phi$ of $\Gamma$,* $\mathrm{Tree}\langle\Phi\rangle \subseteq \mathrm{Tree}\langle\Gamma\rangle$.

**Convention 7.** Let $u$ be a bit string and $\Gamma$ any run. Then $\Gamma^{\preceq u}$ will stand for the result of first removing from $\Gamma$ all labmoves except those that look like $\wp w.\alpha$ for some bit string $w$ with $w \preceq u$, and then deleting this sort of prefixes '$w$.' in the remaining labmoves, i.e. replacing each remaining labmove $\wp w.\alpha$ (where $w$ is a bit string) by $\wp\alpha$. Example: If $u = 101000\ldots$ and $\Gamma = \langle \top\epsilon.\alpha_1, \bot:, \bot 1.\alpha_2, \top 0.\alpha_3, \bot 1:, \top 10.\alpha_4\rangle$, then $\Gamma^{\preceq u} = \langle \top\alpha_1, \bot\alpha_2, \top\alpha_4\rangle$.

**Definition 8.** *Let $A$ be a constant game. The game $\mathbin{\downarrow\!\!\circ}A$ is defined by:*

1. *A position* $\Phi$ *is in* $\mathbf{Lr}^{\mathfrak{b}A}$ *iff* $\Phi$ *is prelegal and, for every leaf* $w$ *of* $\mathrm{Tree}\langle\Phi\rangle$, $\Phi^{\preceq w} \in \mathbf{Lr}^{A}$.
2. *As long as* $\Gamma \in \mathbf{Lr}^{\mathfrak{b}A}$, $\mathbf{Wn}^{\mathfrak{b}A}\langle\Gamma\rangle = \top$ *iff* $\mathbf{Wn}^{A}\langle\Gamma^{\preceq u}\rangle = \top$ *for every infinite bit string* $u$.[4]

*Next, we officially reiterate the earlier-given definition of* $\circ\!\!-$ *by stipulating that* $A \circ\!\!- B =_{def} \mathfrak{b}A \rightarrow B$.

**Remark 9.** Intuitively, a legal run $\Gamma$ of $\mathfrak{b}A$ can be thought of as a multiset $Z$ of parallel legal runs of $A$. Specifically, $Z = \{\Gamma^{\preceq u} \mid u \textit{ is a complete branch of } \textit{Tree}\langle\Gamma\rangle\}$, with complete branches of $\textit{Tree}\langle\Gamma\rangle$ thus acting as names for — or "representing" — elements of $Z$. In order for $\top$ to win, every run from $Z$ should be a $\top$-won run of $A$. The runs from $Z$ typically share some common initial segments and, put together, can be seen as forming a tree of labmoves, with $\textit{Tree}\langle\Gamma\rangle$ — that we call the *underlying tree-structure* of $Z$ — in a sense presenting the shape of that tree. The meaning of a replicative move $w$: — making which is an exclusive privilege of $\bot$ — is creating in (the evolving) $Z$ two copies of position $\Gamma^{\preceq w}$ out of one. And the meaning of a nonreplicative move $w.\alpha$ is making move $\alpha$ in all positions $\Gamma^{\preceq u}$ of (the evolving) $Z$ with $w \preceq u$. This is a brutally brief explanation, of course. The reader may find it very helpful to see Section 13 of [7] for detailed explanations and illustrations of the intuitions associated with our $\mathfrak{b}$-related formal concepts.[5]

# 4 Not-necessarily-constant games

Constant games can be seen as generalized propositions: while propositions in classical logic are just elements of $\{\top, \bot\}$, constant games are functions from runs to $\{\top, \bot\}$. As we know, however, propositions only offer a very limited expressive power, and classical logic needs to consider the more general concept of predicates, with propositions being nothing but special — constant — cases of predicates. The situation in CL is similar. Our concept of (simply) game generalizes that of constant game in the same sense as the classical concept of predicate generalizes that of proposition.

Let us fix two infinite sets of expressions: the set $\{v_1, v_2, \ldots\}$ of **variables** and the set $\{1, 2, \ldots\}$ of **constants**. Without loss of generality here we assume that the above collection of constants is exactly the universe of discourse — i.e. the set over which the variables range — in all cases that we consider. By a **valuation** we mean a function $e$ that sends each variable $x$ to a constant $e(x)$. In these terms, a classical predicate $P$ can be understood as a function that sends each valuation $e$ to a proposition, i.e. constant predicate. Similarly, what we call a game sends valuations to constant games:

[4] For reasons pointed out on page 39 of [7], the phrase "for every infinite bit string $u$" here can be equivalently replaced by "for every complete branch $u$ of $\textit{Tree}\langle\Gamma\rangle$". Similarly, in clause 1, "every leaf $w$ of $\textit{Tree}\langle\Phi\rangle$" can be replaced by "every infinite bit string $w$".

[5] A couple of potentially misleading typos have been found in Section 13 of [7]. The current erratum note is maintained at http://www.csc.villanova.edu/~japaridz/CL/erratum.pdf

**Definition 10.** *A* **game** *is a function $A$ from valuations to constant games. We write $e[A]$ (rather than $A(e)$) to denote the constant game returned by $A$ for valuation $e$. Such a constant game $e[A]$ is said to be an* **instance** *of $A$. For readability, we often write $\mathbf{Lr}^A_e$ and $\mathbf{Wn}^A_e$ instead of $\mathbf{Lr}^{e[A]}$ and $\mathbf{Wn}^{e[A]}$.*

Just as this is the case with propositions versus predicates, constant games in the sense of Definition 2 will be thought of as special, constant cases of games in the sense of Definition 10. In particular, each constant game $A'$ is the game $A$ such that, for every valuation $e$, $e[A] = A'$. From now on we will no longer distinguish between such $A$ and $A'$, so that, if $A$ is a constant game, it is its own instance, with $A = e[A]$ for every $e$.

We say that a game $A$ **depends on** a variable $x$ iff there are two valuations $e_1, e_2$ that agree on all variables except $x$ such that $e_1[A] \neq e_2[A]$. Constant games thus do not depend on any variables.

Just as the Boolean operations straightforwardly extend from propositions to all predicates, our operations $\neg, \wedge, \vee, \rightarrow, \sqcap, \sqcup, \mathbin{\dot{\wedge}}, \multimap$ extend from constant games to all games. This is done by simply stipulating that $e[\ldots]$ commutes with all of those operations: $\neg A$ is the game such that, for every $e$, $e[\neg A] = \neg e[A]$; $A \sqcap B$ is the game such that, for every $e$, $e[A \sqcap B] = e[A] \sqcap e[B]$; etc.

To generalize the standard operation of substitution of variables to games, let us agree that by a **term** we mean either a variable or a constant; the domain of each valuation $e$ is extended to all terms by stipulating that, for any constant $c$, $e(c) = c$.

**Definition 11.** *Let $A$ be a game, $x_1, \ldots, x_n$ pairwise distinct variables, and $t_1, \ldots, t_n$ any (not necessarily distinct) terms. The result of* **substituting** *$x_1, \ldots, x_n$* **by** *$t_1, \ldots, t_n$* **in** *$A$, denoted $A(x_1/t_1, \ldots, x_n/t_n)$, is defined by stipulating that, for every valuation $e$, $e[A(x_1/t_1, \ldots, x_n/t_n)] = e'[A]$, where $e'$ is the valuation for which we have $e'(x_1) = e(t_1)$, $\ldots$, $e'(x_n) = e(t_n)$ and, for every variable $y \notin \{x_1, \ldots, x_n\}$, $e'(y) = e(y)$.*

Intuitively $A(x_1/t_1, \ldots, x_n/t_n)$ is $A$ with $x_1, \ldots, x_n$ remapped to $t_1, \ldots, t_n$, respectively. Following the standard readability-improving practice established in the literature for predicates, we will often fix a tuple $(x_1, \ldots, x_n)$ of pairwise distinct variables for a game $A$ and write $A$ as $A(x_1, \ldots, x_n)$. It should be noted that when doing so, by no means do we imply that $x_1, \ldots, x_n$ are of all of (or only) the variables on which $A$ depends. Representing $A$ in the form $A(x_1, \ldots, x_n)$ sets a context in which we can write $A(t_1, \ldots, t_n)$ to mean the same as the more clumsy expression $A(x_1/t_1, \ldots, x_n/t_n)$.

In the above terms, we now officially reiterate the earlier-given definitions of the two main quantifier-style operations $\bigsqcap$ and $\bigsqcup$:

$$\bigsqcap x A(x) =_{def} A(1) \sqcap A(2) \sqcap A(3) \sqcap \ldots$$

and

$$\bigsqcup x A(x) =_{def} A(1) \sqcup A(2) \sqcup A(3) \sqcup \ldots.$$

# 5 Computational problems and their algorithmic solvability

Our games are obviously general enough to model anything that one would call a (two-agent, two-outcome) interactive problem. However, they are a bit too general. There are games where the chances of a player to succeed essentially depend on the relative speed at which its adversary acts. A simple example would be a game where both players have a legal move in the initial position, and which is won by the player who moves first. CL does not want to consider this sort of games meaningful computational problems. Definition 4.2 of [7] imposes a simple condition on games and calls games satisfying that condition **static**. We are not reproducing that definition here as it is not relevant for our purposes. It should be however mentioned that, according to one of the theses on which the philosophy of CL relies, the concept of static games is an adequate formal counterpart of our intuitive concept of "pure", speed-independent interactive computational problems. All meaningful and reasonable examples of games — including all elementary games — are static, and the class of static games is closed under all of the game operations that we have seen (Theorem 14.1 of [7]). Let us agree that from now on the term "**computational problem**", or simply "**problem**", is a synonym of "static game".

Now it is time to explain what *computability* of such problems means. The definitions given in this section are semiformal. The omitted technical details are rather standard or irrelevant and can be easily restored by anyone familiar with Turing machines. If necessary, the corresponding detailed definitions can be found in Part II of [7].

[7] defines two models of interactive computation, called the *hard-play machine* (*HPM*) and the *easy-play machine* (*EPM*). Both are sorts of Turing machines with the capability of making moves, and have three tapes: the ordinary read/write *work tape*, and the read-only *valuation* and *run* tapes. The valuation tape contains a full description of some valuation $e$ (say, by listing the values of $e$ at $v_1, v_2, \ldots$), and its content remains fixed throughout the work of the machine. As for the run tape, it serves as a dynamic input, at any time spelling the current position, i.e. the sequence of the (lab)moves made by the two players so far. So, every time one of the players makes a move, that move — with the corresponding label — is automatically appended to the content of this tape. In the HPM model, there is no restriction on the frequency of environment's moves. In the EPM model, on the other hand, the machine has full control over the speed of its adversary: the environment can (but is not obligated to) make a (one single) move only when the machine explicitly allows it to do so — the event that we call **granting permission**. The only "fairness" requirement that such a machine is expected to satisfy is that it should grant permission every once in a while; how long that "while" lasts, however, is totally up to the machine. The HPM and EPM models seem to be two extremes, yet, according to Theorem 17.2 of [7], they yield the same class of winnable static games. The present paper will only deal with the EPM model, so let us take a little closer look at it.

Let $\mathcal{M}$ be an EPM. A *configuration* of $\mathcal{M}$ is defined in the standard way: this is a full description of the ("current") state of the machine, the locations of its three scanning heads and the contents of its tapes, with the exception that, in order to make finite descriptions of configurations possible, we do not formally include a description of the unchanging (and possibly essentially infinite) content of the valuation tape as a part of configuration, but rather account for it in our definition of computation branch as will be seen below. The *initial configuration* is the configuration where $\mathcal{M}$ is in its start state and the work and run tapes are empty. A configuration $C'$ is said to be an *e-successor* of configuration $C$ in $\mathcal{M}$ if, when valuation $e$ is spelled on the valuation tape, $C'$ can legally follow $C$ in the standard — standard for multitape Turing machines — sense, based on the transition function (which we assume to be deterministic) of the machine and accounting for the possibility of nondeterministic updates — depending on what move $\bot$ makes or whether it makes a move at all — of the content of the run tape when $\mathcal{M}$ grants permission. Technically granting permission happens by entering one of the specially designated states called "permission states". An **e-computation branch** of $\mathcal{M}$ is a sequence of configurations of $\mathcal{M}$ where the first configuration is the initial configuration and every other configuration is an $e$-successor of the previous one. Thus, the set of all $e$-computation branches captures all possible scenarios (on valuation $e$) corresponding to different behaviors by $\bot$. Such a branch is said to be **fair** iff permission is granted infinitely many times in it. Each $e$-computation branch $B$ of $\mathcal{M}$ incrementally spells — in the obvious sense — a run $\Gamma$ on the run tape, which we call the **run spelled by** $B$. Then, for a game $A$ we write $\mathcal{M} \models_e A$ to mean that, whenever $\Gamma$ is the run spelled by some $e$-computation branch $B$ of $\mathcal{M}$ and $\Gamma$ is not $\bot$-illegal, then branch $B$ is fair and $\mathbf{Wn}_e^A\langle\Gamma\rangle = \top$. We write $\mathcal{M} \models A$ and say that $\mathcal{M}$ **computes (solves, wins)** $A$ iff $\mathcal{M} \models_e A$ for every valuation $e$. Finally, we write $\models A$ and say that $A$ is **computable** iff there is an EPM $\mathcal{M}$ with $\mathcal{M} \models A$.

# 6 The language of INT and the extended language

As mentioned, the language of intuitionistic logic can be seen as a fragment of that of CL. The main building blocks of the language of **INT** are infinitely many **problem letters**, or **letters** for short, for which we use $P, Q, R, S, \ldots$ as metavariables. They are what in classical logic are called 'predicate letters', and what CL calls 'general letters'. With each letter is associated a nonnegative integer called its *arity*. \$ is one of the letters, with arity 0. We refer to it as the **logical letter**, and call all other letters **nonlogical**. The language also contains infinitely many *variables* and *constants* — exactly the ones fixed in Section 4. "*Term*" also has the same meaning as before. An **atom** is $P(t_1, \ldots, t_n)$, where $P$ is an $n$-ary letter and the $t_i$ are terms. Such an atom is said to be *P-based*. If here each term $t_i$ is a constant, we say that $P(t_1, \ldots, t_n)$ is **grounded**. A $P$-based atom is $n$-ary, logical, nonlogical etc. iff $P$ is so. When $P$ is 0-ary, we write $P$ instead of $P()$. **INT-Formulas** are the elements of the smallest class of expressions such that all

atoms are **INT**-formulas and, if $F, F_1, \ldots, F_n$ ($n \geq 2$) are **INT**-formulas and $x$ is a variable, then the following expressions are also **INT**-formulas: $(F_1) \circ\!\!- (F_2)$, $(F_1) \sqcap \ldots \sqcap (F_n)$, $(F_1) \sqcup \ldots \sqcup (F_n)$, $\sqcap x(F)$, $\sqcup x(F)$. Officially there is no negation in the language of **INT**. Rather, the intuitionistic negation of $F$ is understood as $F \circ\!\!- \$$. In this paper we also employ a more expressive formalism that we call the **extended language**. The latter has the additional connectives $\top, \bot, \neg, \wedge, \vee, \rightarrow, \mathbin{!}$ on top of those of the language of **INT**, extending the above formation rules by adding the clause that $\top$, $\bot$, $\neg F$, $(F_1) \wedge \ldots \wedge (F_n)$, $(F_1) \vee \ldots \vee (F_n)$, $(F_1) \rightarrow (F_2)$ and $\mathbin{!}(F)$ are formulas as long as $F$, $F_1, \ldots, F_n$ are so. $\top$ and $\bot$ count as logical atoms. Henceforth by (simply) "**formula**", unless otherwise specified, we mean a formula of the extended language. Parentheses will often be omitted in formulas when this causes no ambiguity. With $\sqcap$ and $\sqcup$ being **quantifiers**, the definitions of **free** and **bound** occurrences of variables are standard.

In concordance with a similar notational convention for games on which we agreed in Section 4, sometimes a formula $F$ will be represented as $F(x_1, \ldots, x_n)$, where the $x_i$ are pairwise distinct variables. When doing so, we do not necessarily mean that each such $x_i$ has a free occurrence in $F$, or that every variable occurring free in $F$ is among $x_1, \ldots, x_n$. In the context set by the above representation, $F(t_1, \ldots, t_n)$ will mean the result of replacing, in $F$, each free occurrence of each $x_i$ ($1 \leq i \leq n$) by term $t_i$. In case each $t_i$ is a variable $y_i$, it may be not clear whether $F(x_1, \ldots, x_n)$ or $F(y_1, \ldots, y_n)$ was originally meant to represent $F$ in a given context. Our disambiguating convention is that the context is set by the expression that was used earlier. That is, when we first mention $F(x_1, \ldots, x_n)$ and only after that use the expression $F(y_1, \ldots, y_n)$, the latter should be understood as the result of replacing variables in the former rather than vice versa.

Let $x$ be a variable, $t$ a term and $F(x)$ a formula. $t$ is said to be **free for $x$ in** $F(x)$ iff none of the free occurrences of $x$ in $F(x)$ is in the scope of $\sqcap t$ or $\sqcup t$. Of course, when $t$ is a constant, this condition is always satisfied.

An **interpretation** is a function $^*$ that sends each $n$-ary letter $P$ to a static game $^*P = P^*(x_1, \ldots, x_n)$, where the $x_i$ are pairwise distinct variables. This function induces a unique mapping that sends each formula $F$ to a game $F^*$ (in which case we say that $^*$ **interprets** $F$ as $F^*$ and that $F^*$ is the **interpretation of** $F$ under $^*$) by stipulating that:

1. Where $P$ is an $n$-ary letter with $^*P = P^*(x_1, \ldots, x_n)$ and $t_1, \ldots, t_n$ are terms, $(P(t_1, \ldots, t_n))^* = P^*(t_1, \ldots, t_n)$.
2. $^*$ commutes with all operators: $\top^* = \top$, $(F \circ\!\!- G)^* = F^* \circ\!\!- G^*$, $(F_1 \wedge \ldots \wedge F_n)^* = F_1^* \wedge \ldots \wedge F_n^*$, $(\sqcap x F)^* = \sqcap x(F^*)$, etc.

When a given formula is represented as $F(x_1, \ldots, x_n)$, we will typically write $F^*(x_1, \ldots, x_n)$ instead of $\big(F(x_1, \ldots, x_n)\big)^*$.

For a formula $F$, we say that an interpretation $^*$ is **admissible for** $F$, or simply **$F$-admissible**, iff the following conditions are satisfied:

1. For every $n$-ary letter $P$ occurring in $F$, where $^*P = P^*(x_1, \ldots, x_n)$, the game $P^*(x_1, \ldots, x_n)$ does not depend on any variables that are not among $x_1, \ldots, x_n$ but occur (whether free or bound) in $F$.

2. $\$^* = B \sqcap F_1^* \sqcap F_2^* \sqcap \ldots$, where $B$ is an arbitrary problem and $F_1, F_2, \ldots$ is the lexicographic list of all grounded nonlogical atoms of the language.

Speaking philosophically, an admissible interpretation $^*$ interprets \$ as a "strongest problem": the interpretation of every grounded atom and hence — according to Lemma 27 — of every formula is reducible to $\$^*$, and reducible in a certain uniform, interpretation-independent way. Viewing $\$^*$ as a resource, it can be seen as a universal resource that allows its owner to solve any problem. Our choice of the dollar notation here is no accident: money is an illustrative example of an all-powerful resource in the world where everything can be bought. "Strongest", "universal" or "all-powerful" do not necessarily mean "impossible". So, the intuitionistic negation $F \circ\!\!- \$$ of $F$ here does not have the traditional "$F^*$ is absurd" meaning. Rather, it means that $F^*$ (too) is of universal strength. Turing completeness, NP-completeness and similar concepts are good examples of "being of universal strength". $\$^*$ is what [7] calls a *standard universal problem* of the universe $\langle F_1^*, F_2^*, \ldots\rangle$. Briefly, a *universal problem* of a universe (sequence) $\langle A_1, A_2, \ldots\rangle$ of problems is a problem $U$ such that $\models U \rightarrow A_1 \sqcap A_2 \sqcap \ldots$ and hence $\models U \circ\!\!- A_1 \sqcap A_2 \sqcap \ldots$, intuitively meaning a problem to which each $A_i$ is reducible. For every $B$, the problem $U = B \sqcap A_1 \sqcap A_2 \ldots$ satisfies this condition, and universal problems of this particular form are called *standard*. Every universal problem $U$ of a given universe can be shown to be equivalent to a standard universal problem $U'$ of the same universe, in the sense that $\models U \circ\!\!- U'$ and $\models U' \circ\!\!- U$. And all of the operators of **INT** can be shown to preserve equivalence. Hence, restricting universal problems to standard ones does not result in any loss of generality: a universal problem can always be safely assumed to be standard. See section 23 of [7] for an extended discussion of the philosophy and intuitions associated with universal problems. Here we only note that interpreting \$ as a universal problem rather than (as one might expect) as $\bot$ yields more generality, for $\bot$ is nothing but a special, extreme case of a universal problem. *Our soundness theorem for* **INT**, *of course, continues to hold with* $\bot$ *instead of* \$.

Let $F$ be a formula. We write $\Vdash F$ and say that $F$ is **valid** iff $\models F^*$ for every $F$-admissible interpretation $^*$. For an EPM $\mathcal{E}$, we write $\mathcal{E} \Vdash F$ and say that $\mathcal{E}$ is a **uniform solution** for $F$ iff $\mathcal{E} \models F^*$ for every $F$-admissible interpretation $^*$. Finally, we write $\Vdash\!\!\!\Vdash F$ and say that $F$ is **uniformly valid** iff there is a uniform solution for $F$. Note that uniform validity automatically implies validity but not vice versa. Yet, these two concepts have been conjectured to be extensionally equivalent (Conjecture 26.2 of [7]).

# 7 The Gentzen-style axiomatization of INT

A **sequent** is a pair $\underline{G} \Rightarrow K$, where $K$ is an **INT**-formula and $\underline{G}$ is a (possibly empty) finite sequence of **INT**-formulas. In what follows, $E, F, K$ will be used as metavariables for formulas, and $\underline{G}, \underline{H}$ as metavariables for sequences of formulas. We think of sequents as formulas, identifying $\Rightarrow K$ with $K$, $F \Rightarrow K$ with $\mathbin{!}F \rightarrow K$

(i.e. $F \circ\!\!- K$), and $E_1, \ldots, E_n \Rightarrow K$ $(n \geq 2)$ with $\mathbin{!}\!\circ E_1 \wedge \ldots \wedge \mathbin{!}\!\circ E_n \rightarrow K$.[6] This allows us to automatically extend the concepts such as validity, uniform validity, etc. from formulas to sequents. A formula $K$ is considered **provable** in **INT** iff the sequent $\Rightarrow K$ is so.

Deductively, logic **INT** is given by the following 15 rules. This axiomatization is known (or can be easily shown) to be equivalent to other "standard" formulations, including Hilbert-style axiomatizations for **INT** and/or versions where a primitive symbol for negation is present while \$ is absent, or where $\sqcap$ and $\sqcup$ are strictly binary, or where variables are the only terms of the language.[7]

Below $\underline{G}, \underline{H}$ are any (possibly empty) sequences of **INT**-formulas; $n \geq 2$; $1 \leq i \leq n$; $x$ is any variable; $E$, $F$, $K$, $F_1$, ..., $F_n$, $K_1$, ..., $K_n$, $F(x)$, $K(x)$ are any **INT**-formulas; $y$ is any variable not occurring (whether free or bound) in the conclusion of the rule; in Left $\bigsqcap$ (resp. Right $\bigsqcup$), $t$ is any term free for $x$ in $F(x)$ (resp. in $K(x)$). $\mathcal{P} \longmapsto C$ means "from premise(s) $\mathcal{P}$ conclude $C$". When there are multiple premises in $\mathcal{P}$, they are separated with a semicolon.

| | | | |
|---|---|---|---|
| **Identity** | | $\longmapsto$ | $K \Rightarrow K$ |
| **Domination** | | $\longmapsto$ | $\$ \Rightarrow K$ |
| **Exchange** | $\underline{G}, E, F, \underline{H} \Rightarrow K$ | $\longmapsto$ | $\underline{G}, F, E, \underline{H} \Rightarrow K$ |
| **Weakening** | $\underline{G} \Rightarrow K$ | $\longmapsto$ | $\underline{G}, F \Rightarrow K$ |
| **Contraction** | $\underline{G}, F, F \Rightarrow K$ | $\longmapsto$ | $\underline{G}, F \Rightarrow K$ |
| **Right** $\circ\!\!-$ | $\underline{G}, F \Rightarrow K$ | $\longmapsto$ | $\underline{G} \Rightarrow F \circ\!\!- K$ |
| **Left** $\circ\!\!-$ | $\underline{G}, F \Rightarrow K_1;\ \underline{H} \Rightarrow K_2$ | $\longmapsto$ | $\underline{G}, \underline{H}, K_2 \circ\!\!- F \Rightarrow K_1$ |
| **Right** $\sqcap$ | $\underline{G} \Rightarrow K_1;\ \ldots;\ \underline{G} \Rightarrow K_n$ | $\longmapsto$ | $\underline{G} \Rightarrow K_1 \sqcap \ldots \sqcap K_n$ |
| **Left** $\sqcap$ | $\underline{G}, F_i \Rightarrow K$ | $\longmapsto$ | $\underline{G}, F_1 \sqcap \ldots \sqcap F_n \Rightarrow K$ |
| **Right** $\sqcup$ | $\underline{G} \Rightarrow K_i$ | $\longmapsto$ | $\underline{G} \Rightarrow K_1 \sqcup \ldots \sqcup K_n$ |
| **Left** $\sqcup$ | $\underline{G}, F_1 \Rightarrow K;\ \ldots;\ \underline{G}, F_n \Rightarrow K$ | $\longmapsto$ | $\underline{G}, F_1 \sqcup \ldots \sqcup F_n \Rightarrow K$ |
| **Right** $\bigsqcap$ | $\underline{G} \Rightarrow K(y)$ | $\longmapsto$ | $\underline{G} \Rightarrow \bigsqcap x K(x)$ |
| **Left** $\bigsqcap$ | $\underline{G}, F(t) \Rightarrow K$ | $\longmapsto$ | $\underline{G}, \bigsqcap x F(x) \Rightarrow K$ |
| **Right** $\bigsqcup$ | $\underline{G} \Rightarrow K(t)$ | $\longmapsto$ | $\underline{G} \Rightarrow \bigsqcup x K(x)$ |
| **Left** $\bigsqcup$ | $\underline{G}, F(y) \Rightarrow K$ | $\longmapsto$ | $\underline{G}, \bigsqcup x F(x) \Rightarrow K$ |

[6] In order to fully remain within the language of **INT**, we could understand $E_1, \ldots, E_n \Rightarrow K$ as $E_1 \circ\!\!- (E_2 \circ\!\!- \ldots \circ\!\!- (E_n \circ\!\!- K)\ldots)$, which can be shown to be equivalent to our present understanding. We, however, prefer to read $E_1, \ldots, E_n \Rightarrow K$ as $\mathbin{!}\!\circ E_1 \wedge \ldots \wedge \mathbin{!}\!\circ E_n \rightarrow K$ as it seems more convenient to work with.

[7] That we allow constants is only for technical convenience. This does not really yield a stronger language, as constants behave like free variables and can be thought of as such.

**Theorem 12.** **(Soundness:)** *If* **INT** $\vdash S$, *then* $\Vdash S$ *(any sequent* $S$*). Furthermore,* **(uniform-constructive soundness:)** *there is an effective procedure that takes any* **INT***-proof of any sequent* $S$ *and constructs a uniform solution for* $S$.

*Proof.* See Section 12. □

# 8 CL2-derived validity lemmas

In our proof of Theorem 12 we will need a number of lemmas concerning uniform validity of certain formulas. Some such validity proofs will be given directly in Section 10. But some proofs come "for free", based on the soundness theorem for logic **CL2** proven in [12]. **CL2** is a propositional logic whose logical atoms are $\top$ and $\bot$ (but not \$) and whose connectives are $\neg, \wedge, \vee, \rightarrow, \sqcap, \sqcup$. It has two sorts of nonlogical atoms, called *elementary* and *general*. General atoms are nothing but 0-ary atoms of our extended language; elementary atoms, however, are something not present in the extended language. We refer to the formulas of the language of **CL2** as **CL2-formulas**. In this paper, the **CL2**-formulas that do not contain elementary atoms — including $\top$ and $\bot$ that count as such — are said to be **general-base**. Thus, every general-base formula is a formula of our extended language, and its validity or uniform validity means the same as in Section 6.[8]

Understanding $F \rightarrow G$ as an abbreviation for $\neg F \vee G$, a **positive occurrence** in a **CL2**-formula is an occurrence that is in the scope of an even number of occurrences of $\neg$; otherwise the occurrence is **negative**. A **surface occurrence** is an occurrence that is not in the scope of $\sqcap$ and/or $\sqcup$. The **elementarization** of a **CL2**-formula $F$ is the result of replacing in $F$ every surface occurrence of every subformula of the form $G_1 \sqcap \ldots \sqcap G_n$ (resp. $G_1 \sqcup \ldots \sqcup G_n$) by $\top$ (resp. $\bot$), and every positive (resp. negative) surface occurrence of every general atom by $\bot$ (resp. $\top$). A **CL2**-formula $F$ is said to be **stable** iff its elementarization is a tautology of classical logic. With these conventions, **CL2** is axiomatized by the following three rules:

(a) $\vec{H} \longmapsto F$, where $F$ is stable and $\vec{H}$ is the smallest set of formulas such that, whenever $F$ has a positive (resp. negative) surface occurrence of a subformula $G_1 \sqcap \ldots \sqcap G_n$ (resp. $G_1 \sqcup \ldots \sqcup G_n$), for each $i \in \{1, \ldots, n\}$, $\vec{H}$ contains the result of replacing that occurrence in $F$ by $G_i$.

(b) $H \longmapsto F$, where $H$ is the result of replacing in $F$ a negative (resp. positive) surface occurrence of a subformula $G_1 \sqcap \ldots \sqcap G_n$ (resp. $G_1 \sqcup \ldots \sqcup G_n$) by $G_i$ for some $i \in \{1, \ldots, n\}$.

(c) $H \longmapsto F$, where $H$ is the result of replacing in $F$ two — one positive and one negative — surface occurrences of some general atom by a nonlogical elementary atom that does not occur in $F$.

[8] These concepts extend to the full language of **CL2** as well, with interpretations required to send elementary atoms to elementary games (i.e. predicates in the classical sense, understood in CL as games that have no nonemty legal runs).

In this section $p, q, r, s, t, u, w \ldots$ (possibly with indices) will exlusively stand for nonlogical elementary atoms, and $P, Q, R, S, T, U, W$ (possibly with indices) stand for general atoms. All of these atoms are implicitely assumed to be pairwise distinct in each context.

**Convention 13.** In Section 7 we started using the notation $\underline{G}$ for sequences of formulas. Later we agreed to identify sequences of formulas with $\wedge$-conjunctions of those formulas. So, from now on, an underlined expression such as $\underline{G}$ will mean an arbitrary formula $G_1 \wedge \ldots \wedge G_n$ for some $n \geq 0$. Such an expression will always occur in a bigger context such as $\underline{G} \wedge F$ or $\underline{G} \rightarrow F$; our convention is that, when $n = 0$, $\underline{G} \wedge F$ and $\underline{G} \rightarrow F$ simply mean $F$.

As we agreed that $p, q, \ldots$ stand for elementary atoms and $P, Q, \ldots$ for general atoms, $\underline{p}, \underline{q}, \ldots$ will correspondingly mean $\wedge$-conjunctions of elementary atoms, and $\underline{P}, \underline{Q}, \ldots$ mean $\wedge$-conjunctions of general atoms.

We will also be underlining complex expressions such as $F \rightarrow G$, $\sqcap x F(x)$ or $\downarrow F$. $\underline{F \rightarrow G}$ should be understood as $(F_1 \rightarrow G_1) \wedge \ldots \wedge (F_n \rightarrow G_n)$, $\underline{\sqcap x F(x)}$ as $\sqcap x F_1(x) \wedge \ldots \wedge \sqcap x F_n(x)$ (note that only the $F_i$ vary but not $x$), $\underline{\downarrow F}$ as $\downarrow F_1 \wedge \ldots \downarrow F_n$, $\underline{\downarrow\downarrow F}$ as $\downarrow(\downarrow F_1 \wedge \ldots \wedge \downarrow F_n)$, $\underline{\downarrow F} \rightarrow \underline{F} \wedge G$ as $\downarrow F_1 \wedge \ldots \downarrow F_n \rightarrow F_1 \wedge \ldots \wedge F_n \wedge G$, etc.

The axiomatization of **CL2** is rather unusual, but it is easy to get a syntactic feel of it once we do a couple of exercises.

**Example 14.** The following is a **CL2**-proof of $(P \rightarrow Q) \wedge (Q \rightarrow T) \rightarrow (P \rightarrow T)$:

1. $(p \rightarrow q) \wedge (q \rightarrow t) \rightarrow (p \rightarrow t)$ (from $\{\}$ by Rule **(a)**).
2. $(P \rightarrow q) \wedge (q \rightarrow t) \rightarrow (P \rightarrow t)$ (from 1 by Rule **(c)**).
3. $(P \rightarrow Q) \wedge (Q \rightarrow t) \rightarrow (P \rightarrow t)$ (from 2 by Rule **(c)**).
4. $(P \rightarrow Q) \wedge (Q \rightarrow T) \rightarrow (P \rightarrow T)$ (from 3 by Rule **(c)**).

**Example 15.** Let $n \geq 2$, and let $m$ be the length (number of conjuncts) of both $\underline{R}$ and $\underline{r}$.

a) For $i \in \{1, \ldots, n\}$, the formula of Lemma 17(j) is provable in **CL2**. It follows from $(\underline{R} \rightarrow S_i) \rightarrow (\underline{R} \rightarrow S_i)$ by Rule **(b)**; the latter follows from $(\underline{R} \rightarrow s_i) \rightarrow (\underline{R} \rightarrow s_i)$ by Rule **(c)**; the latter, in turn, can be derived from $(\underline{r} \rightarrow s_i) \rightarrow (\underline{r} \rightarrow s_i)$ applying Rule **(c)** $m$ times. Finally, $(\underline{r} \rightarrow s_i) \rightarrow (\underline{r} \rightarrow s_i)$ is its own elementarization and is a classical tautology, so it follows from the empty set of premises by Rule **(a)**.

b) The formula of Lemma 17(h) is also provable in **CL2**. It is derivable by Rule **(a)** from the set of $n$ premises, easch premise being $(\underline{R} \rightarrow S_1) \wedge \ldots \wedge (\underline{R} \rightarrow S_n) \rightarrow (\underline{R} \rightarrow S_i)$ for some $i \in \{1, \ldots, n\}$. The latter is derivable by Rule **(c)** from $(\underline{R} \rightarrow S_1) \wedge \ldots \wedge (\underline{R} \rightarrow s_i) \wedge \ldots \wedge (\underline{R} \rightarrow S_n) \rightarrow (\underline{R} \rightarrow s_i)$. The latter, in turn, can be derived from $(\underline{R} \rightarrow S_1) \wedge \ldots \wedge (\underline{r} \rightarrow s_i) \wedge \ldots \wedge (\underline{R} \rightarrow S_n) \rightarrow (\underline{r} \rightarrow s_i)$ applying Rule **(c)** $m$ times. Finally, the latter follows from the empty set of premises by Rule **(a)**.

Obviously **CL2** is decidable. This logic has been proven sound and complete in [12]. We only need the soundness part of that theorem restricted to general-base formulas. It sounds as follows:

**Lemma 16.** *Any general-base* **CL2***-provable formula is valid. Furthermore, there is an effective procedure that takes any* **CL2***-proof of any such formula $F$ and constructs a uniform solution for $F$.*

A *substitution* is a function $f$ that sends every general atom $P$ of the language of **CL2** to a formula $f(P)$ of the extended language. This mapping extends to all general-base **CL2**-formulas by stipulating that $f$ commutes with all operators: $f(G_1 \to G_2) = f(G_1) \to f(G_2)$, $f(G_1 \sqcap \ldots \sqcap G_k) = f(G_1) \sqcap \ldots \sqcap f(G_k)$, etc. We say that a formula $G$ is a **substitutional instance** of a general-base **CL2**-formula $F$ iff $G = f(F)$ for some substitution $f$. Thus, "$G$ is a substitutional instance of $F$" means that $G$ has the same form as $F$.

In the following lemma, we assume $n \geq 2$ (clauses (h),(i),(j)), and $1 \leq i \leq n$ (clause (j)). Notice that, for the exception of clause (g), the expressions given below are schemata of formulas rather than formulas, for the lengths of their underlined expressions — as well as $i$ and $n$ — may vary.

**Lemma 17.** *All substitutional instances of all formulas given by the following schemata are uniformly valid. Furthermore, there is an effective procedure that takes any particular formula matching a given scheme and constructs an EPM that is a uniform solution for all substitutional instances of that formula.*

*a)* $(\underline{R} \wedge P \wedge Q \wedge \underline{S} \to T) \to (\underline{R} \wedge Q \wedge P \wedge \underline{S} \to T)$;
*b)* $(\underline{R} \to T) \to (\underline{R} \wedge P \to T)$;
*c)* $(\underline{R \to S}) \to (\underline{W} \wedge \underline{R} \wedge \underline{U} \to \underline{W} \wedge \underline{S} \wedge \underline{U})$;
*d)* $\big(\underline{R} \wedge P \to Q\big) \to \big(\underline{R} \to (P \to Q)\big)$;
*e)* $\big(P \to (Q \to T)\big) \wedge (\underline{R} \to Q) \to \big(P \to (\underline{R} \to T)\big)$;
*f)* $\big(P \to (\underline{R} \to Q)\big) \wedge (\underline{S} \wedge Q \to T) \to (\underline{S} \wedge \underline{R} \wedge P \to T)$;
*g)* $(P \to Q) \wedge (Q \to T) \to (P \to T)$;
*h)* $(\underline{R} \to S_1) \wedge \ldots \wedge (\underline{R} \to S_n) \to (\underline{R} \to S_1 \sqcap \ldots \sqcap S_n)$;
*i)* $(\underline{R} \wedge S_1 \to T) \wedge \ldots \wedge (\underline{R} \wedge S_n \to T) \to \big((\underline{R} \wedge (S_1 \sqcup \ldots \sqcup S_n) \to T\big)$;
*j)* $(\underline{R} \to S_i) \to (\underline{R} \to S_1 \sqcup \ldots \sqcup S_n)$.

*Proof.* In order to prove this lemma, it would be sufficient to show that all formulas given by the above schemata are provable in **CL2**. Indeed, if we succeed in doing so, then an effective procedure whose existence is claimed in the lemma could be designed to work as follows: First, the procedure finds a **CL2**-proof of a given formula $F$. Then, based on that proof and using the procedure whose existence is stated in Lemma 16, it finds a uniform solution $\mathcal{E}$ for that formula. It is not hard to see that the same $\mathcal{E}$ will automatically be a uniform solution for every substitutional instance of $F$ as well.

The **CL2**-provability of the formulas given by clauses (g), (h) and (j) has been verified in Examples 14 and 15. A similar verification for the other clauses is left as an easy syntactic exercise for the reader. □

# 9 Closure lemmas

In this section we let $n$ range over natural numbers (including 0), $x$ over any variables, $F, E, G$ (possibly with subscripts) over any formulas, and $\mathcal{E}$, $\mathcal{C}$, $\mathcal{D}$ (possibly with subscripts) over any EPMs. Unless otherwise specified, these metavariables are assumed to be universally quantified in a context. The expression $\{EPMs\}$ stands for the set of all EPMs.

**Lemma 18.** *If $\Vdash F$, then $\Vdash \mathbin{⫰} F$. Furthermore, there is an effective function $h$ : $\{EPMs\} \longrightarrow \{EPMs\}$ such that, for any $\mathcal{E}$ and $F$, if $\mathcal{E} \Vdash F$, then $h(\mathcal{E}) \Vdash \mathbin{⫰} F$.*

*Proof.* According to Proposition 21.2 of [7], there is an effective function $h$ : $\{\text{EPMs}\} \longrightarrow \{\text{EPMs}\}$ such that, for any EPM $\mathcal{E}$ and any static game $A$, if $\mathcal{E} \models A$, then $h(\mathcal{E}) \models \mathbin{⫰} A$. We now claim that if $\mathcal{E} \Vdash F$, then $h(\mathcal{E}) \Vdash \mathbin{⫰} F$. Indeed, assume $\mathcal{E} \Vdash F$. Consider any $\mathbin{⫰} F$-admissible interpretation $^*$. Of course, the same interpretation is also $F$-admissible. Hence, $\mathcal{E} \Vdash F$ implies $\mathcal{E} \models F^*$. But then, by the known behavior of $h$, we have $h(\mathcal{E}) \models \mathbin{⫰} F^*$. Since $^*$ was arbitrary, we conclude that $h(\mathcal{E}) \Vdash \mathbin{⫰} F$. □

**Lemma 19.** *If $\Vdash F$, then $\Vdash \bigsqcap x F$. Furthermore, there is an effective function $h$ : $\{EPMs\} \longrightarrow \{EPMs\}$ such that, for any $\mathcal{E}$, $x$ and $F$, if $\mathcal{E} \Vdash F$, then $h(\mathcal{E}) \Vdash \bigsqcap x F$.*

*Proof.* Similar to the previous lemma, only based on Proposition 21.1 of [7] instead of 21.2. □

**Lemma 20.** *If $\Vdash F$ and $\Vdash F \rightarrow E$, then $\Vdash E$. Furthermore, there is an effective function $h$ : $\{EPMs\} \times \{EPMs\} \longrightarrow \{EPMs\}$ such that, for any $\mathcal{E}$, $\mathcal{C}$, $F$ and $E$, if $\mathcal{E} \Vdash F$ and $\mathcal{C} \Vdash F \rightarrow E$, then $h(\mathcal{E}, \mathcal{C}) \Vdash E$.*

*Proof.* According to Proposition 21.3 of [7], there is an effective function $g$ that takes any HPM $\mathcal{H}$ and EPM $\mathcal{E}$ and returns an EPM $\mathcal{C}$ such that, for any static games $A$,$B$ and any valuation $e$, if $\mathcal{H} \models_e A$ and $\mathcal{E} \models_e A \rightarrow B$, then $\mathcal{C} \models_e B$. Theorem 17.2 of [7], which establishes equivalence between EPMs and HPMs in a constructive sense, allows us to assume that $\mathcal{H}$ is an EPM rather than HPM here. More precisely, we can routinely convert $g$ into an (again effective) function $h$ : $\{\text{EPMs}\} \times \{\text{EPMs}\} \longrightarrow \{\text{EPMs}\}$ such that, for any static games $A$,$B$, valuation $e$ and EPMs $\mathcal{E}$ and $\mathcal{C}$,

$$\text{if } \mathcal{E} \models_e A \text{ and } \mathcal{C} \models_e A \rightarrow B, \text{ then } h(\mathcal{E}, \mathcal{C}) \models_e B. \tag{3}$$

We claim that the above function $h$ behaves as our lemma states. Assume $\mathcal{E} \Vdash F$ and $\mathcal{C} \Vdash F \rightarrow E$, and consider an arbitrary valuation $e$ and an arbitrary $E$-admissible interpretation $^*$. Our goal is to show that $h(\mathcal{E}, \mathcal{C}) \models_e E^*$, which obviously means the same as

$$h(\mathcal{E}, \mathcal{C}) \models_e e[E^*]. \tag{4}$$

We define the new interpretation $^\dagger$ by stipulating that, for every $n$-ary letter $P$ with $P^* = P^*(x_1, \ldots, x_n)$, $P^\dagger$ is the game $P^\dagger(x_1, \ldots, x_n)$ such that, for any tuple $c_1, \ldots, c_n$ of constants, $P^\dagger(c_1, \ldots, c_n) = e[P^*(c_1, \ldots, c_n)]$. Unlike $P^*(x_1, \ldots, x_n)$

that may depend on some "hidden" variables (those that are not among $x_1, \ldots, x_n$), obviously $P^\dagger(x_1, \ldots, x_n)$ does not depend on any variables other that $x_1, \ldots, x_n$. This makes $^\dagger$ admissible for any formula, including $F$ and $F \to E$. Then our assumptions $\mathcal{E}\Vdash F$ and $C\Vdash F \to E$ imply $\mathcal{E} \models_e F^\dagger$ and $C \models_e F^\dagger \to E^\dagger$. Consequently, by (3), $h(\mathcal{E}, C) \models_e E^\dagger$, i.e. $h(\mathcal{E}, C) \models_e e[E^\dagger]$. Now, with some thought, we can see that $e[E^\dagger] = e[E^*]$. Hence (4) is true. □

**Lemma 21. (Modus ponens)** *If* $\Vdash F_1, \ldots, \Vdash F_n$ *and* $\Vdash F_1 \wedge \ldots \wedge F_n \to E$, *then* $\Vdash E$. *Furthermore, there is an effective function* $h : \{EPMs\}^{n+1} \longrightarrow \{EPMs\}$ *such that, for any EPMs* $\mathcal{E}_1, \ldots, \mathcal{E}_n$, $C$ *and any formulas* $F_1, \ldots, F_n, E$, *if* $\mathcal{E}_1\Vdash F_1, \ldots, \mathcal{E}_n\Vdash F_n$ *and* $C\Vdash F_1 \wedge \ldots \wedge F_n \to E$, *then* $h(\mathcal{E}_1, \ldots, \mathcal{E}_n, C)\Vdash E$. *Such a function, in turn, can be effectively constructed for each particular* $n$.

*Proof.* In case $n = 0$, $h$ is simply the identity function $h(C) = C$. In case $n = 1$, $h$ is the function whose existence is stated in Lemma 20. Below we will construct $h$ for case $n = 2$. It should be clear how to generalize that construction to any greater $n$.

Assume $\mathcal{E}_1\Vdash F_1$, $\mathcal{E}_2\Vdash F_2$ and $C\Vdash F_1 \wedge F_2 \to E$. By Lemma 17(d), $(F_1 \wedge F_2 \to E) \to (F_1 \to (F_2 \to E))$ has a uniform solution. Lemma 20 allows us to combine that solution with $C$ and find a uniform solution $\mathcal{D}_1$ for $F_1 \to (F_2 \to E)$. Now applying Lemma 20 to $\mathcal{E}_1$ and $\mathcal{D}_1$, we find a uniform solution $\mathcal{D}_2$ for $F_2 \to E$. Finally, applying the same lemma to $\mathcal{E}_2$ and $\mathcal{D}_2$, we find a uniform solution $\mathcal{D}$ for $E$. Note that $\mathcal{D}$ does not depend on $F_1, F_2, E$, and that we constructed $\mathcal{D}$ in an effective way from the arbitrary $\mathcal{E}_1$, $\mathcal{E}_2$ and $C$. Formalizing this construction yields function $h$ whose existence is claimed by the lemma. □

**Lemma 22. (Transitivity)** *If* $\Vdash F \to E$ *and* $\Vdash E \to G$, *then* $\Vdash F \to G$. *Furthermore, there is an effective function* $h : \{EPMs\} \times \{EPMs\} \longrightarrow \{EPMs\}$ *such that, for any* $\mathcal{E}_1$, $\mathcal{E}_2$, $F$, $E$ *and* $G$, *if* $\mathcal{E}_1\Vdash F \to E$ *and* $\mathcal{E}_2\Vdash E \to G$, *then* $h(\mathcal{E}_1, \mathcal{E}_2)\Vdash F \to G$.

*Proof.* Assume $\mathcal{E}_1\Vdash F \to E$ and $\mathcal{E}_2\Vdash E \to G$. By Lemma 17(g), we also have $C\Vdash (F \to E) \wedge (E \to G) \to (F \to G)$ for some (fixed) $C$. Lemma 21 allows us to combine the three uniform solutions and construct a uniform solution $\mathcal{D}$ for $F \to G$. □

# 10 More validity lemmas

As pointed out in Remark 16.4 of [7], when trying to show that a given EPM wins a given game, it is always safe to assume that the runs that the machine generates are never $\bot$-illegal, i.e. that the environment never makes an illegal move, for if it does, the machine automatically wins. This assumption, that we call the **clean environment assumption**, will always be implicitly present in our winnability proofs.

We will often employ a uniform solution for $P \to P$ called the **copy-cat strategy** (*CCS*). This strategy, that we already saw in Section 2, consists in mimicking, in the antecedent, the moves made by the environment in the consequent, and vice

versa. More formally, the algorithm that $CCS$ follows is an infinite loop, on every iteration of which $CCS$ keeps granting permission until the environment makes a move $1.\alpha$ (resp. $2.\alpha$), to which the machine responds by the move $2.\alpha$ (resp. $1.\alpha$). As shown in the proof of Proposition 22.1 of [7], this strategy guarantees success in every game of the form $A \vee \neg A$ and hence $A \rightarrow A$. An important detail is that $CCS$ never looks at the past history of the game, i.e. the movement of its scanning head on the run tape is exclusively left-to-right. This guarantees that, even if the original game was something else and it only evolved to $A \rightarrow A$ later as a result of making a series of moves, switching to the $CCS$ after the game has been brought down to $A \rightarrow A$ guarantees success no matter what happened in the past.

Thgroughout this section, $F$, $G$, $E$, $K$ (possibly with indices and attached tuples of variables) range over formulas, $x$ and $y$ over variables, $t$ over terms, $n$ over nonnegative integers, $w$ over bit strings, and $\alpha, \gamma$ over moves. These (meta)variables are assumed to be universally quantified in a context unless otherwise specified. In accordance with our earlier convention, $\epsilon$ means the empty string, so that, say, '$1.\epsilon.\alpha$' is the same as '$1..\alpha$'.

**Lemma 23.** $\mathbb{H}\!\!-\!\!\downarrow\!\!\circ F \rightarrow F$. *Furthermore, there is an EPM $\mathcal{E}$ such that, for any $F$, $\mathcal{E}\mathbb{H}\!\!-\!\!\downarrow\!\!\circ F \rightarrow F$.*

*Proof.* The idea of a uniform solution $\mathcal{E}$ for $\downarrow\!\!\circ F \rightarrow F$ is very simple: just act as $CCS$, never making any replicative moves in the antecedent and pretending that the latter is $F$ rather than (the stronger) $\downarrow\!\!\circ F$. The following formal description of the interactive algorithm that $\mathcal{E}$ follows is virtually the same as that of $CCS$, with the only difference that the move prefix '1.' is replaced by '$1.\epsilon.$' everywhere.

**Procedure** LOOP: Keep granting permission until the environment makes a move $1.\epsilon.\alpha$ or $2.\alpha$; in the former case make the move $2.\alpha$, and in the latter case make the move $1.\epsilon.\alpha$; then repeat LOOP.

Fix an arbitrary valuation $e$, interpretation $*$ and $e$-computation branch $B$ of $\mathcal{E}$. Let $\Theta$ be the run spelled by $B$. From the description of LOOP it is clear that permission will be granted infinitely many times in $B$, so this branch is fair. Hence, in order to show that $\mathcal{E}$ wins the game, it would suffice to show that $\mathbf{Wn}_e^{\downarrow\!\!\circ F^* \rightarrow F^*}\langle\Theta\rangle = \top$.

Let $\Theta_i$ denote the initial segment of $\Theta$ consisting of the (lab)moves made by the players by the beginning of the $i$th iteration of LOOP in $B$ (if such an iteration exists). By induction on $i$, based on the clean environment assumption and applying a routine analysis of the the behavior of LOOP and the definitions of the relevant game operations, one can easily find that

a) $\Theta_i \in \mathbf{Lr}_e^{\downarrow\!\!\circ F^* \rightarrow F^*}$;
b) $\Theta_i^{1.\epsilon.} = \neg\Theta_i^{2.}$;
c) *All of the moves in $\Theta_i^{1.}$ have the prefix '$\epsilon.$'.*

If LOOP is iterated infinitely many times, then the above obviously extends from $\Theta_i$ to $\Theta$, because every initial segment of $\Theta$ is an initial segment of some $\Theta_i$. And if LOOP is iterated only a finite number $m$ of times, then $\Theta = \Theta_m$. This

is so because the environment cannot make a move $1.\epsilon.\alpha$ or $2.\alpha$ during the $m$th iteration (otherwise there would be a next iteration), and any other move would violate the clean environment assumption; and, as long as the environment does not move during a given iteration, neither does the machine. Thus, no matter whether LOOP is iterated a finite or infinite number of times, we have:

$$\begin{array}{ll} \text{a) } \Theta \in \mathbf{Lr}_e^{\mathfrak{b}F^*\to F^*}; & \\ \text{b) } \Theta^{1.\epsilon.} = \neg\Theta^{2.}\,; & (5) \\ \text{c) } \textit{All of the moves in } \Theta^{1.} \textit{ have the prefix '}\epsilon.\textit{'}. & \end{array}$$

Since $\Theta \in \mathbf{Lr}_e^{\mathfrak{b}F^*\to F^*}$, in order to show that $\mathbf{Wn}_e^{\mathfrak{b}F^*\to F^*}\langle\Theta\rangle = \top$, by the definition of $\to$, it would suffice to show that either $\mathbf{Wn}_e^{F^*}\langle\Theta^{2.}\rangle = \top$ or $\mathbf{Wn}_e^{\neg\mathfrak{b}F^*}\langle\Theta^{1.}\rangle = \top$. So, assume $\mathbf{Wn}_e^{F^*}\langle\Theta^{2.}\rangle \neq \top$, i.e. $\mathbf{Wn}_e^{F^*}\langle\Theta^{2.}\rangle = \bot$, i.e. $\mathbf{Wn}_e^{\neg F^*}\langle\neg\Theta^{2.}\rangle = \top$. Then, by clause (b) of (5), $\mathbf{Wn}_e^{\neg F^*}\langle\Theta^{1.\epsilon.}\rangle = \top$, i.e. $\mathbf{Wn}_e^{F^*}\langle\neg\Theta^{1.\epsilon.}\rangle = \bot$, i.e. $\mathbf{Wn}_e^{F^*}\langle(\neg\Theta^{1.})^{\epsilon.}\rangle = \bot$. Pick any infinite bit string $w$. In view of clause (c) of (5), we obviously have $(\neg\Theta^{1.})^{\epsilon.} = (\neg\Theta^{1.})^{\preceq w}$. Hence $\mathbf{Wn}_e^{F^*}\langle(\neg\Theta^{1.})^{\preceq w}\rangle = \bot$. But this, by the definition of $\mathfrak{b}$, implies $\mathbf{Wn}_e^{\mathfrak{b}F^*}\langle\neg\Theta^{1.}\rangle = \bot$. The latter, in turn, can be rewritten as the desired $\mathbf{Wn}_e^{\neg\mathfrak{b}F^*}\langle\Theta^{1.}\rangle = \top$.

Thus, we have shown that $\mathcal{E}$ wins $\mathfrak{b}F^* \to F^*$. Since $^*$ was arbitrary and the work of $\mathcal{E}$ did not depend on it, we conclude that $\mathcal{E} \Vdash \mathfrak{b}F \to F$. □

In the subsequent constructions found in this section, $^*$ will always mean an arbitrary but fixed interpretation admissible for the formula whose uniform validity we are trying to prove. Next, $e$ will always mean an arbitrary but fixed valuation — specifically, the valuation spelled on the valuation tape of the machine under question. For readability, we will usually omit the $e$ parameter when it is irrelevant. Also, having already seen one example, in the remaining uniform validity proofs we will typically limit ourselves to just constructing interactive algorithms, leaving the (usually routine) verification of their correctness to the reader. An exception will be the proof of Lemma 34 where, due to the special complexity of the case, correctness verification will be done even more rigorously than we did this in the proof of Lemma 23.

**Lemma 24.** $\Vdash \mathfrak{b}(F \to G) \to (\mathfrak{b}F \to \mathfrak{b}G)$. *Moreover, there is an EPM $\mathcal{E}$ such that, for every $F$ and $G$, $\mathcal{E} \Vdash \mathfrak{b}(F \to G) \to (\mathfrak{b}F \to \mathfrak{b}G)$.*

*Proof.* A relaxed description of a uniform solution $\mathcal{E}$ for $\mathfrak{b}(F \to G) \to (\mathfrak{b}F \to \mathfrak{b}G)$ is as follows. In $\mathfrak{b}(F^* \to G^*)$ and $\mathfrak{b}F^*$ the machine is making exactly the same replicative moves (moves of the form $w$:) as the environment is making in $\mathfrak{b}G^*$. This ensures that the tree-structures of the three $\mathfrak{b}$-components of the game are identical, and now all the machine needs for a success is to win the game $(F^* \to G^*) \to (F^* \to G^*)$ within each branch of those trees. This can be easily achieved by applying copy-cat methods to the two occurrences of $F$ and the two occurrences of $G$.

In precise terms, the strategy that $\mathcal{E}$ follows is described by the following interactive algorithm.

**Procedure** LOOP: Keep granting permission until the adversary makes a move $\gamma$. Then:
If $\gamma = 2.2.w$:, then make the moves $1.w$: and $2.1.w$:, and repeat LOOP;
If $\gamma = 2.2.w.\alpha$ (resp. $\gamma = 1.w.2.\alpha$), then make the move $1.w.2.\alpha$ (resp. $2.2.w.\alpha$), and repeat LOOP;
If $\gamma = 2.1.w.\alpha$ (resp. $\gamma = 1.w.1.\alpha$), then make the move $1.w.1.\alpha$ (resp. $2.1.w.\alpha$), and repeat LOOP. □

**Lemma 25.** $\Vdash \downarrow\!\circ F_1 \wedge \ldots \wedge \downarrow\!\circ F_n \rightarrow \downarrow\!\circ(F_1 \wedge \ldots \wedge F_n)$. *Furthermore, there is an effective procedure that takes any particular value of $n$ and constructs an EPM $\mathcal{E}$ such that, for any $F_1, \ldots, F_n$, $\mathcal{E} \Vdash \downarrow\!\circ F_1 \wedge \ldots \wedge \downarrow\!\circ F_n \rightarrow \downarrow\!\circ(F_1 \wedge \ldots \wedge F_n)$.*

*Proof.* The idea of a uniform solution here is rather similar to the one in the proof of Lemma 24. Here is the algorithm:

**Procedure** LOOP: Keep granting permission until the adversary makes a move $\gamma$. Then:
If $\gamma = 2.w$:, then make the $n$ moves $1.1.w$:, $\ldots$, $1.n.w$:, and repeat LOOP;
If $\gamma = 2.w.i.\alpha$ (resp. $\gamma = 1.i.w.\alpha$) where $1 \leq i \leq n$, then make the move $1.i.w.\alpha$ (resp. $2.w.i.\alpha$), and repeat LOOP. □

**Lemma 26.** $\Vdash \downarrow\!\circ F \rightarrow \downarrow\!\circ F \wedge \downarrow\!\circ F$. *Furthermore, there is an EPM $\mathcal{E}$ such that, for any $F$, $\mathcal{E} \Vdash \downarrow\!\circ F \rightarrow \downarrow\!\circ F \wedge \downarrow\!\circ F$.*

*Proof.* The idea of a winning strategy here is to first replicate the antecedent turning it into something "essentially the same"[9] as $\downarrow\!\circ F^* \wedge \downarrow\!\circ F^*$, and then switch to a strategy that is "essentially the same as" the ordinary copy-cat strategy. Precisely, here is how $\mathcal{E}$ works: it makes the move $1.\epsilon$: (replicating the antecedent), after which it follows the following algorithm:

**Procedure** LOOP: Keep granting permission until the adversary makes a move $\gamma$. Then:
If $\gamma = 1.0\alpha$ (resp. $\gamma = 2.1.\alpha$), then make the move $2.1.\alpha$ (resp. $1.0\alpha$), and repeat LOOP;
If $\gamma = 1.1\alpha$ (resp. $\gamma = 2.2.\alpha$), then make the move $2.2.\alpha$ (resp. $1.1\alpha$), and repeat LOOP;
If $\gamma = 1.\epsilon.\alpha$, then make the moves $2.1.\epsilon.\alpha$ and $2.2.\epsilon.\alpha$, and repeat LOOP. □

Remember from Section 4 that, when $t$ is a constant, $e(t) = t$.

**Lemma 27.** *For any* **INT**-*formula $K$, $\Vdash \downarrow\!\circ\$ \rightarrow K$. Furthermore, there is an effective procedure that takes any* **INT**-*formula $K$ and constructs a uniform solution for $\downarrow\!\circ\$ \rightarrow K$.*

[9] Using the notation $\circ$ introduced in Section 13 of [7], in precise terms this "something" is $\downarrow\!\circ(F^* \circ F^*)$.

*Proof.* We construct an EPM $\mathcal{E}$ and verify that it is a uniform solution for $\mathbin{⅋}\$ \rightarrow K$; both the construction and the verification will be done by induction on the complexity of $K$. The goal in each case is to show that $\mathcal{E}$ generates a $\top$-won run of $e[(\mathbin{⅋}\$ \rightarrow K)^*] = \mathbin{⅋}e[\$^*] \rightarrow e[K^*]$ which, according to our convention, we may write with "$e$" omitted.

*Case 1:* $K = \$$. This case is taken care of by Lemma 23.

*Case 2:* $K$ is a $k$-ary nonlogical atom $P(t_1, \ldots, t_k)$. Let $c_1, \ldots, c_k$ be the constants $e(t_1), \ldots, e(t_k)$, respectively. Evidently in this case $e[K^*] = e[P^*(c_1, \ldots, c_k)]$, so, the game for which $\mathcal{E}$ needs to generate a winning run is $\mathbin{⅋}\$^* \rightarrow P^*(c_1, \ldots, c_k)$. Assume $(P(c_1, \ldots, c_k))^*$ is conjunct $\#m$ of (the infinite $\sqcap$-conjunction) $\$^*$. We define $\mathcal{E}$ to be the EPM that acts as follows. At the beginning, if necessary (i.e. unless all $t_i$ are constants), it reads the valuation tape to find $c_1, \ldots, c_k$. Then, using this information, it finds the above number $m$ and makes the move '$1.\epsilon.m$', which can be seen[10] to bring the game down to $\mathbin{⅋}P^*(c_1, \ldots, c_k) \rightarrow P^*(c_1, \ldots, c_k)$. After this move, $\mathcal{E}$ switches to the strategy whose existence is stated in Lemma 23.

*Case 3:* $K = \mathbin{⅋}E \rightarrow F$. By Lemma 17(b), there is a uniform solution for $(\mathbin{⅋}\$ \rightarrow F) \rightarrow (\mathbin{⅋}\$ \wedge \mathbin{⅋}E \rightarrow F)$. Lemmas 17(d) and 22 allow us to convert the latter into a uniform solution $\mathcal{D}$ for $(\mathbin{⅋}\$ \rightarrow F) \rightarrow \big(\mathbin{⅋}\$ \rightarrow (\mathbin{⅋}E \rightarrow F)\big)$. By the induction hypothesis, there is also a uniform solution $\mathcal{C}$ for $\mathbin{⅋}\$ \rightarrow F$. Applying Lemma 21 to $\mathcal{C}$ and $\mathcal{D}$, we find a uniform solution $\mathcal{E}$ for $\mathbin{⅋}\$ \rightarrow (\mathbin{⅋}E \rightarrow F)$.

*Case 4:* $K = E_1 \sqcup \ldots \sqcup E_n$. By the induction hypothesis, we know how to construct an EPM $\mathcal{C}_1$ with $\mathcal{C}_1 \Vdash \mathbin{⅋}\$ \rightarrow E_1$. Now we define $\mathcal{E}$ to be the EPM that first makes the move 2.1, and then plays the rest of the game as $\mathcal{C}_1$ would play. $\mathcal{E}$ can be seen to be successful because its initial move 2.1 brings $(\mathbin{⅋}\$ \rightarrow K)^*$ down to $(\mathbin{⅋}\$ \rightarrow E_1)^*$.

*Case 5:* $K = E_1 \sqcap \ldots \sqcap E_n$. By the induction hypothesis, for each $i$ with $1 \leq i \leq n$ we have an EPM $\mathcal{C}_i$ with $\mathcal{C}_i \Vdash \mathbin{⅋}\$ \rightarrow E_i$. We define $\mathcal{E}$ to be the EPM that acts as follows. At the beginning, $\mathcal{E}$ keeps granting permission until the adversary makes a move. The clean environment assumption guarantees that this move should be $2.i$ for some $1 \leq i \leq n$. It brings $(\mathbin{⅋}\$ \rightarrow E_1 \sqcap \ldots \sqcap E_n)^*$ down to $(\mathbin{⅋}\$ \rightarrow E_i)^*$. If and after such a move $2.i$ is made, $\mathcal{E}$ continues the rest of the play as $\mathcal{C}_i$.

*Case 6:* $K = \sqcup x E(x)$. By the induction hypothesis, there is an EPM $\mathcal{C}_1$ with $\mathcal{C}_1 \Vdash \mathbin{⅋}\$ \rightarrow E(1)$. Now we define $\mathcal{E}$ to be the EPM that first makes the move 2.1, and then plays the rest of the game as $\mathcal{C}_1$. $\mathcal{E}$ can be seen to be successful because its initial move 2.1 brings $\big(\mathbin{⅋}\$ \rightarrow \sqcup x E(x)\big)^*$ down to $\big(\mathbin{⅋}\$ \rightarrow E(1)\big)^*$.

*Case 7:* $K = \sqcap x E(x)$. By the induction hypothesis, for each constant $c$ there is (and can be effectively found) an EPM $\mathcal{C}_c$ with $\mathcal{C}_c \Vdash \mathbin{⅋}\$ \rightarrow E(c)$. Now we define $\mathcal{E}$ to be the EPM that acts as follows. At the beginning, $\mathcal{E}$ keeps granting permission until the adversary makes a move. By the clean environment assumption, such a move should be $2.c$ for some constant $c$. This move can be seen to bring $\big((\mathbin{⅋}\$ \rightarrow \sqcap x E(x))\big)^*$ down to $\big(\mathbin{⅋}\$ \rightarrow E(c)\big)^*$. If and after the above move $2.c$ is made, $\mathcal{E}$ plays the rest of the game as $\mathcal{C}_c$. □

[10] See Proposition 13.8 of [7].

**Lemma 28.** *Assume $n \geq 2$, $1 \leq i \leq n$, and $t$ is a term free for $x$ in $G(x)$. Then the following uniform validities hold. Furthermore, in each case there is an effective procedure that takes any particular values of $n$, $i$, $t$ and constructs an EPM which is a uniform solution for the corresponding formula no matter what the values of $F_1, \ldots, F_n$ and/or $G(x)$ (as long as $t$ is free for $x$ in $G(x)$) are.*

*a)* $\Vdash \mathbin{!}\circ(F_1 \sqcap \ldots \sqcap F_n) \rightarrow \mathbin{!}\circ F_i$;
*b)* $\Vdash \mathbin{!}\circ \bigsqcap x G(x) \rightarrow \mathbin{!}\circ G(t)$;
*c)* $\Vdash \mathbin{!}\circ(F_1 \sqcup \ldots \sqcup F_n) \rightarrow \mathbin{!}\circ F_1 \sqcup \ldots \sqcup \mathbin{!}\circ F_n$;
*d)* $\Vdash \mathbin{!}\circ \bigsqcup x G(x) \rightarrow \bigsqcup x \mathbin{!}\circ G(x)$.

*Proof.* Below come winning strategies for each case.
*Clause (a):* Make the move '$1.\epsilon.i$'. This brings the game down to $\mathbin{!}\circ F_i^* \rightarrow \mathbin{!}\circ F_i^*$. Then switch to *CCS*.
*Clause (b):* Let $c = e(t)$. Read $c$ from the valuation tape if necessary, i.e., if $t$ is a variable (otherwise, simply $c = t$). Then make the move '$1.\epsilon.c$'. This brings the game down to $\mathbin{!}\circ G^*(c) \rightarrow \mathbin{!}\circ G^*(c)$. Now, switch to *CCS*.
*Clause (c):* Keep granting permission until the adversary makes a move '$1.\epsilon.j$' ($1 \leq j \leq n$), bringing the game down to $\mathbin{!}\circ F_j^* \rightarrow \mathbin{!}\circ F_1^* \sqcup \ldots \sqcup \mathbin{!}\circ F_n^*$. If and after such a move is made (and if not, a win is automatically guaranteed), make the move '$2.j$', which brings the game down to $\mathbin{!}\circ F_j^* \rightarrow \mathbin{!}\circ F_j^*$. Finally, switch to *CCS*.
*Clause (d):* Keep granting permission until the adversary makes the move '$1.\epsilon.c$' for some constant $c$. This brings the game down to $\mathbin{!}\circ G^*(c) \rightarrow \bigsqcup x \mathbin{!}\circ G^*(x)$. Now make the move '$2.c$', which brings the game down to $\mathbin{!}\circ G^*(c) \rightarrow \mathbin{!}\circ G^*(c)$. Finally, switch to *CCS*. □

**Lemma 29.** $\Vdash \bigsqcap x\big(F(x) \rightarrow G(x)\big) \rightarrow \big(\bigsqcap x F(x) \rightarrow \bigsqcap x G(x)\big)$. *Furthermore, there is an EPM $\mathcal{E}$ such that, for any $F(x)$ and $G(x)$, $\mathcal{E} \Vdash \bigsqcap x\big(F(x) \rightarrow G(x)\big) \rightarrow \big(\bigsqcap x F(x) \rightarrow \bigsqcap x G(x)\big)$.*

*Proof.* Strategy: Wait till the environment makes the move '$2.2.c$' for some constant $c$. This brings the $\bigsqcap x G^*(x)$ component down to $G^*(c)$ and hence the entire game to $\bigsqcap x\big(F^*(x) \rightarrow G^*(x)\big) \rightarrow \big(\bigsqcap x F^*(x) \rightarrow G^*(c)\big)$. Then make the same move $c$ in the antecedent and in $\bigsqcap x F^*(x)$, i.e. make the moves '$1.c$' and '$2.1.c$'. The game will be brought down to $\big(F^*(c) \rightarrow G^*(c)\big) \rightarrow \big(F^*(c) \rightarrow G^*(c)\big)$. Finally, switch to *CCS*. □

**Lemma 30.** $\Vdash \bigsqcap x\big(F_1(x) \wedge \ldots \wedge F_n(x) \wedge E(x) \rightarrow G(x)\big) \rightarrow \big(\bigsqcap x F_1(x) \wedge \ldots \wedge \bigsqcap x F_n(x) \wedge \bigsqcup x E(x) \rightarrow \bigsqcup x G(x)\big)$. *Furthermore, there is an effective procedure that takes any particular value of $n$ and constructs an EPM $\mathcal{E}$ such that, for any $F_1(x)$, ..., $F_n(x)$, $E(x)$, $G(x)$, $\mathcal{E} \Vdash \bigsqcap x\big(F_1(x) \wedge \ldots \wedge F_n(x) \wedge E(x) \rightarrow G(x)\big) \rightarrow \big(\bigsqcap x F_1(x) \wedge \ldots \wedge \bigsqcap x F_n(x) \wedge \bigsqcup x E(x) \rightarrow \bigsqcup x G(x)\big)$.*

*Proof.* Strategy for $n$: Wait till the environment makes a move $c$ in the $\bigsqcup x E^*(x)$ component. Then make the same move $c$ in $\bigsqcup x G^*(x)$, $\bigsqcap x\big(F_1^*(x) \wedge \ldots \wedge F_n^*(x) \wedge E^*(x) \rightarrow G^*(x)\big)$ and each of the $\bigsqcap x F_i^*(x)$ components. After this series of moves the game will have evolved to $\big(F_1^*(c) \wedge \ldots \wedge F_n^*(c) \wedge E^*(c) \rightarrow G^*(c)\big) \rightarrow \big(F_1^*(c) \wedge \ldots \wedge F_n^*(c) \wedge E^*(c) \rightarrow G^*(c)\big)$. Now switch to *CCS*. □

**Lemma 31.** *Assume $t$ is free for $x$ in $F(x)$. Then $\mathbin{⫦}\!\!-F(t) \to \bigsqcup xF(x)$. Furthermore, there is an effective function that takes any $t$ and constructs an EPM $\mathcal{E}$ such that, for any $F(x)$, whenever $t$ is free for $x$ in $F(x)$, $\mathcal{E}\mathbin{⫦}\!\!-F(t) \to \bigsqcup xF(x)$.*

*Proof.* Strategy: Let $c = e(t)$. Read $c$ from the valuation tape if necessary. Then make the move '2.$c$', bringing the game down to $F^*(c) \to F^*(c)$. Then switch to *CCS*. □

**Lemma 32.** *Assume $F$ does not contain $x$. Then $\mathbin{⫦}\!\!-F \to \bigsqcap xF$. Furthermore, there is an EPM $\mathcal{E}$ such that, for any $F$ and $x$, as long as $F$ does not contain $x$, $\mathcal{E}\mathbin{⫦}\!\!-F \to \bigsqcap xF$.*

*Proof.* In this case we prefer to explicitly write the usually suppressed parameter $e$. Consider an arbitrary $F$ not containing $x$, and an arbitrary interpretation $^*$ admissible for $F \to \bigsqcap xF$. The formula $F \to \bigsqcap xF$ contains $x$ yet $F$ does not. From the definition of admissibility and with a little thought we can see that $F^*$ does not depend on $x$. In turn, this means — as can be seen with some thought — that the move $c$ by the environment (whatever constant $c$) in $e[\bigsqcap xF^*]$ brings this game down to $e[F^*]$. With this observation in mind, the following strategy can be seen to be successful: Wait till the environment makes the move '2.$c$' for some constant $c$. Then read the sequence '1.$\alpha_1$', ..., '1.$\alpha_n$' of (legal) moves possibly made by the environment before it made the move '2.$c$', and make the $n$ moves '2.$\alpha_1$', ..., '2.$\alpha_n$'. It can be seen that now the original game $e[F^*] \to e[\bigsqcap xF^*]$ will have been brought down to $\langle\Phi\rangle e[F^*] \to \langle\Phi\rangle e[F^*]$, where $\Phi = \langle \top\alpha_1, \ldots, \top\alpha_n\rangle$. So, switching to *CCS* at this point guarantees success. □

**Lemma 33.** *Assume $F(x)$ does not contain $y$. Then $\mathbin{⫦}\!\!-\bigsqcap yF(y) \to \bigsqcap xF(x)$ and $\mathbin{⫦}\!\!-\bigsqcup xF(x) \to \bigsqcup yF(y)$. In fact, CCS$\mathbin{⫦}\!\!-\bigsqcap yF(y) \to \bigsqcap xF(x)$ and CCS$\mathbin{⫦}\!\!-\bigsqcup xF(x) \to \bigsqcup yF(y)$.*

*Proof.* Assuming that $F(x)$ does not contain $y$ and analyzing the relevant definitions, it is not hard to see that, for any interpretation $^*$ admissible for $\bigsqcap yF(y) \to \bigsqcap xF(x)$ and/or $\bigsqcup xF(x) \to \bigsqcup yF(y)$, we simply have $\big(\bigsqcap yF(y)\big)^* = \big(\bigsqcap xF(x)\big)^*$ and $\big(\bigsqcup yF(y)\big)^* = \big(\bigsqcup xF(x)\big)^*$. So, in both cases we deal with a game of the form $A \to A$, for which the ordinary copy-cat strategy is successful. □

Our proof of the following lemma is fairly long, for which reason it is given separately in Section 11:

**Lemma 34.** *$\mathbin{⫦}\!\!-\mathbin{⫰}F \to \mathbin{⫰}\mathbin{⫰}F$. Furthermore, there is an EPM $\mathcal{E}$ such that, for any $F$, $\mathcal{E}\mathbin{⫦}\!\!-\mathbin{⫰}F \to \mathbin{⫰}\mathbin{⫰}F$.*

# 11 Proof of Lemma 34

Roughly speaking, the uniform solution $\mathcal{E}$ for $\mathbin{⫰}F \to \mathbin{⫰}\mathbin{⫰}F$ that we are going to construct essentially uses a copy-cat strategy between the antecedent and the consequent. However, this strategy cannot be applied directly in the form of our kind

old friend *CCS*. The problem is that the underlying tree-structure (see Remark 9) of the multiset of (legal) runs of $F^*$ that is being generated in the antecedent should be a simple tree $T$, while in the consequent it is in fact what can be called a tree $T''$ of trees. The trick that $\mathcal{E}$ uses is that it sees each edge of $T$ in one of two — blue or yellow — colors. This allows $\mathcal{E}$ to associate with each branch $y$ of a branch $x$ of $T''$ a single branch $z$ of $T$, and vice versa. Specifically, with $x, y, z$ being (encoded by) bit strings, $z$ is such that the subsequence of its blue-colored bits (=edges) coincides with $x$, and the subsequence of its yellow-colored bits coincides with $y$. By appropriately "translating" and "copying" in the antecedent the replicative moves made by the environment in the consequent, such an isomorphism between the branches of $T$ and the branches of branches of $T''$ can be successfully maintained throughout the play. With this one-to-one correspondence in mind, every time the environment makes a (nonreplicative) move in the position(s) of $F^*$ represented by a leaf or a set of leaves of $T$, the machine repeats the same move in the position(s) represented by the corresponding leaf-of-leaf or leaves-of-leaves of $T''$, and vice versa. The effect achieved by this strategy is that the multisets of all runs of $F^*$ in the antecedent and in the consequent of $\mathbin{!}F^* \rightarrow \mathbin{!}\mathbin{!}F^*$ are identical, which, of course, guarantees a win for $\mathcal{E}$.

Let us now define things more precisely. A **colored bit** $b$ is a pair $(c, d)$, where $c$, called the **content** of $b$, is in $\{0, 1\}$, and $d$, called the **color** of $b$, is in $\{blue, yellow\}$. We will be using the notation $\overline{c}$ ("blue $c$") for the colored bit whose content is $c$ and color is *blue*, and $\underline{c}$ ("yellow $c$") for the bit whose content is $c$ and color is *yellow*.

A **colored bit string** is a finite or infinite sequence of colored bits. Consider a colored bit string $v$. The **content** of $v$ is the result of "ignoring the colors" in $v$, i.e. replacing every bit of $v$ by the content of that bit. The **blue content** of $v$ is the content of the string that results from deleting in $v$ all but blue bits. **Yellow content** is defined similarly. We use $\overline{v}$, $\overline{v}$ and $\underline{v}$ to denote the content, blue content and yellow content of $v$, respectively. Example: if $v = \overline{1}\underline{00}\overline{0}\underline{1}$, we have $\overline{v} = 10001$, $\overline{v} = 10$ and $\underline{v} = 001$. As in the case of ordinary bit strings, $\epsilon$ stands for the empty colored bit string. And, for colored bit strings $w$ and $u$, $w \preceq u$ again means that $w$ is a (not necessarily proper) initial segment of $u$.

**Definition 35.** *A* **colored tree** *is a set $T$ of colored bit strings, called its* **branches**, *such that the following conditions are satisfied:*

*a) The set $\{\overline{v} \mid v \in T\}$ — that we denote by $\overline{T}$ — is a tree in the sense of Convention 5.*

*b) For any $w, u \in T$, if $\overline{w} = \overline{u}$, then $w = u$.*

*c) For no $v \in T$ do we have $\{v\overline{0}, v\underline{1}\} \subseteq T$ or $\{v\underline{0}, v\overline{1}\} \subseteq T$.*

*A branch $v$ of $T$ is said to be a* **leaf** *iff $\overline{v}$ is a leaf of $\overline{T}$.*

When represented in the style of Figure 1 of [7] (page 36), a colored tree will look like an ordinary tree, with the only difference that now every edge will have one of the colors *blue* or *yellow*. Also, by condition (c), both of the outgoing edges ("sibling" edges) of any non-leaf node will have the same color.

**Lemma 36.** *Assume $T$ is a colored tree, and $w, u$ are branches of $T$ with $\overline{w} \preceq \overline{u}$ and $\underline{w} \preceq \underline{u}$. Then $w \preceq u$.*

*Proof.* Assume $T$ is a colored tree, $w, u \in T$, and $w \not\preceq u$. We want to show that then $\overline{w} \not\preceq \overline{u}$ or $\underline{w} \not\preceq \underline{u}$. Let $v$ be the longest common initial segment of $w$ and $u$, so we have $w = vw'$ and $u = vu'$ for some $w', u'$ such that $w'$ is nonempty and $w'$ and $u'$ do not have a nonempty common initial segment. Assume the first bit of $w'$ is $\overline{0}$ (the cases when it is $\overline{1}$, $\underline{0}$ or $\underline{1}$, of course, will be similar). If $u'$ is empty, then $w$ obviously contains more blue bits than $u$ does, and we are done. Assume now $u'$ is nonempty, in particular, $b$ is the first bit of $u'$. Since $w'$ and $u'$ do not have a nonempty common initial segment, $b$ should be different from $\overline{0}$. By condition (b) of Definition 35, the content of $b$ cannot be 0 (for otherwise we would have $v\overline{0} = vb$ and hence $b = \overline{0}$). Consequently, $b$ is either $\overline{1}$ or $\underline{1}$. The case $b = \underline{1}$ is ruled out by condition (c) of Definition 35. Thus, $b = \overline{1}$. But the blue content of $v\overline{0}$ is $\overline{v}0$ while the blue content of $v\overline{1}$ is $\overline{v}1$. Taking into account the obvious fact that the former is an initial segment of $\overline{w}$ and the latter is an initial segment of $\overline{u}$, we find $\overline{w} \not\preceq \overline{u}$. □

Now comes a description of our EPM $\mathcal{E}$. At the beginning, this machine creates a record $T$ of the type 'finite colored tree', and initializes it to $\{\epsilon\}$. After that, $\mathcal{E}$ follows the following procedure:

**Procedure** LOOP: Keep granting permission until the adversary makes a move $\gamma$. If $\gamma$ satisfies the conditions of one of the following four cases, act as the corresponding case prescribes. Otherwise go to an infinite loop in a permission state.
*Case (i):* $\gamma = 2.w$: for some bit string $w$. Let $v_1, \ldots, v_k$ be[11] all of the leaves $v$ of $T$ with $w = \overline{v}$. Then make the moves $1.\overline{v}_1:, \ldots, 1.\overline{v}_k:$, update $T$ to $T \cup \{v_1\overline{0}, v_1\overline{1}, \ldots, v_k\overline{0}, v_k\overline{1}\}$, and repeat LOOP.
*Case (ii):* $\gamma = 2.w.u$: for some bit strings $w, u$. Let $v_1, \ldots, v_k$ be all of the leaves $v$ of $T$ such that $w \preceq \overline{v}$ and $u = \underline{v}$. Then make the moves $1.\overline{v}_1:, \ldots, 1.\overline{v}_k:$, update $T$ to $T \cup \{v_1\underline{0}, v_1\underline{1}, \ldots, v_k\underline{0}, v_k\underline{1}\}$, and repeat LOOP.
*Case (iii):* $\gamma = 2.w.u.\alpha$ for some bits strings $w, u$ and move $\alpha$. Let $v_1, \ldots, v_k$ be all of the leaves $v$ of $T$ such that $w \preceq \overline{v}$ and $u \preceq \underline{v}$. Then make the moves $1.\overline{v}_1.\alpha, \ldots, 1.\overline{v}_k.\alpha$, and repeat LOOP.
*Case (iv):* $\gamma = 1.w.\alpha$ for some bit string $w$ and move $\alpha$. Let $v_1, \ldots, v_k$ be all of the leaves $v$ of $T$ with $w \preceq \overline{v}$. Then make the moves $2.\overline{v}_1.\underline{v}_1.\alpha, \ldots, 2.\overline{v}_k.\underline{v}_k.\alpha$, and repeat LOOP.

Fix any interpretation $^*$, valuation $e$ and $e$-computation branch $B$ of $\mathcal{E}$. Let $\Theta$ be the run spelled by $B$. From the above description it is immediately clear that $B$ is a fair. Hence, in order to show that $\mathcal{E}$ wins, it would be sufficient to show that $\mathbf{Wn}_e^{\diamond F^* \to \diamond\diamond F^*}\langle\Theta\rangle = \top$. Notice that the work of $\mathcal{E}$ does not depend on $e$. And, as $e$ is fixed, we can safely and unambiguously omit this parameter (as we often did in the previous section) in expressions such as $e[A]$, $\mathbf{Lr}_e^A$ or $\mathbf{Wn}_e^A$ and just write

[11] In each of the four cases we assume that the list $v_1, \ldots, v_k$ is arranged lexicographically.

or say $A$, $\mathbf{Lr}^A$ or $\mathbf{Wn}^A$. Of course, $\mathcal{E}$ is interpretation-blind, so as long as it wins $\mathbin{⫰}F^* \rightarrow \mathbin{⫰}\mathbin{⫰}F^*$, it is a uniform solution for $\mathbin{⫰}F \rightarrow \mathbin{⫰}\mathbin{⫰}F$.

Let $N = \{1, \ldots, m\}$ if LOOP is iterated the finite number $m$ of times in $B$, and $N = \{1, 2, 3, \ldots\}$ otherwise. For $i \in N$, we let $T_i$ denote the value of record $T$ at the beginning of the $i$th iteration of LOOP; $\Theta_i$ will mean the initial segment of $\Theta$ consisting of the moves made by the beginning of the $i$th iteration of LOOP. Finally, $\Phi_i$ will stand for $\neg\Theta_i^{1\cdot}$ and $\Psi_i$ for $\Theta_i^{2\cdot}$.

From the description of LOOP it is immediately obvious that, for each $i \in N$, $T_i$ is a finite colored tree, and that $T_1 \subseteq T_2 \subseteq \ldots \subseteq T_i$. In our subsequent reasoning we will implicitly rely on this fact.

**Lemma 37.** *For every $i$ with $i \in N$, we have:*

*a)* $\Phi_i$ *is prelegal and* $\mathrm{Tree}\langle\Phi_i\rangle = \underline{\overline{T_i}}$.

*b)* $\Psi_i$ *is prelegal.*

*c) For every leaf $x$ of* $\mathrm{Tree}\langle\Psi_i\rangle$, $\Psi_i^{\preceq x}$ *is prelegal.*

*d) For every leaf $z$ of $T_i$, $\overline{z}$ is a leaf of* $\mathrm{Tree}\langle\Psi_i\rangle$ *and $\underline{z}$ is a leaf of* $\mathrm{Tree}\langle\Psi_i^{\preceq\overline{z}}\rangle$.

*e) For every leaf $x$ of* $\mathrm{Tree}\langle\Psi_i\rangle$ *and every leaf $y$ of* $\mathrm{Tree}\langle\Psi_i^{\preceq x}\rangle$, *there is a leaf $z$ of $T_i$ such that $x = \overline{z}$ and $y = \underline{z}$. By Lemma 36, such a $z$ is unique.*

*f) For every leaf $z$ of $T_i$, $\Phi_i^{\preceq\overline{z}} = (\Psi_i^{\preceq\overline{z}})^{\preceq\underline{z}}$.*

*g) $\Theta_i$ is a legal position of $\mathbin{⫰}F^* \rightarrow \mathbin{⫰}\mathbin{⫰}F^*$; hence, $\Phi_i \in \mathbf{Lr}^{\mathbin{⫰}F^*}$ and $\Psi_i \in \mathbf{Lr}^{\mathbin{⫰}\mathbin{⫰}F^*}$.*

*Proof.* We proceed by induction on $i$. The basis case with $i = 1$ is rather straightforward for each clause of the lemma and we do not discuss it. For the induction step, assume $i + 1 \in N$, and the seven clauses of the lemma are true for $i$.
*Clause (a):* By the induction hypothesis, $\Phi_i$ is prelegal and $\mathit{Tree}\langle\Phi_i\rangle = \underline{\overline{T_i}}$. Assume first that the $i$th iteration of LOOP deals with Case (i), so that $\Phi_{i+1} = \langle\Phi_i, \bot\overline{v}_1 :, \ldots, \bot\overline{v}_k :\rangle$. Each of $\overline{v}_1, \ldots, \overline{v}_k$ is a leaf of $\underline{\overline{T_i}}$, i.e. a leaf of $\mathit{Tree}\langle\Phi_i\rangle$. This guarantees that $\Phi_{i+1}$ is prelegal. Also, by the definition of function *Tree*, we have $\mathit{Tree}\langle\Phi_{i+1}\rangle = \mathit{Tree}\langle\Phi_i\rangle \cup \{\overline{v}_1 0, \overline{v}_1 1, \ldots, \overline{v}_k 0, \overline{v}_k 1\}$. But the latter is nothing but $\underline{\overline{T_{i+1}}}$ as can be seen from the description of how Case (i) updates $T_i$ to $T_{i+1}$. A similar argument applies when the $i$th iteration of LOOP deals with Case (ii). Assume now the $i$th iteration of LOOP deals with Case (iii). Note that the moves made in the antecedent of $\mathbin{⫰}F^* \rightarrow \mathbin{⫰}\mathbin{⫰}F^*$ (the moves that bring $\Phi_i$ to $\Phi_{i+1}$) are nonreplicative — specifically, look like $\overline{v}.\alpha$ where $\overline{v} \in \underline{\overline{T_i}} = \mathit{Tree}\langle\Phi_i\rangle$. Such moves yield prelegal positions and do not change the value of *Tree*, so that $\mathit{Tree}\langle\Phi_i\rangle = \mathit{Tree}\langle\Phi_{i+1}\rangle$. It remains to note that $T$ is not updated in this subcase, so that we also have $\underline{\overline{T_{i+1}}} = \underline{\overline{T_i}}$. Hence $\mathit{Tree}\langle\Phi_{i+1}\rangle = \underline{\overline{T_{i+1}}}$. Finally, suppose the $i$th iteration of LOOP deals with Case (iv). It is the environment who moves in the antecedent of $\mathbin{⫰}F^* \rightarrow \mathbin{⫰}\mathbin{⫰}F^*$, and does so before the machine makes any moves. Then the clean environment assumption — in conjunction with the induction hypothesis — implies that such a move cannot bring $\Phi_i$ to an illegal position of $\mathbin{⫰}F^*$ and hence cannot bring it to a non-prelegal position. So, $\Phi_{i+1}$ is prelegal. As for $\mathit{Tree}\langle\Phi_{i+1}\rangle = \underline{\overline{T_{i+1}}}$, it holds for the same reason as in the previous case.

*Clause (b):* If the $i$th iteration of LOOP deals with Case (i), (ii) or (iii), it is the environment who moves in the consequent of $\mathbin{!}F^* \rightarrow \mathbin{!}\mathbin{!}F^*$, and the clean environment assumption guarantees that $\Psi_{i+1}$ is prelegal. Assume now that the $i$th iteration of LOOP deals with Case (iv), so that $\Psi_{i+1} = \langle \Psi_i, \top\overline{v}_1.\underline{v}_1.\alpha, \ldots, \top\overline{v}_k.\underline{v}_k.\alpha\rangle$. By the induction hypothesis for clause (d), each $\overline{v}_j$ $(1 \leq j \leq k)$ is a leaf of $\mathit{Tree}\langle\Psi_i\rangle$, so adding the moves $\top\overline{v}_1.\underline{v}_1.\alpha, \ldots, \top\overline{v}_k.\underline{v}_k$ does not bring $\Psi_i$ to a non-prelegal position, nor does it modify $\mathit{Tree}\langle\Psi_i\rangle$ because the moves are nonreplicative. Hence $\Psi_{i+1}$ is prelegal.

*Clause (c):* Just as in the previous clause, when the $i$th iteration of LOOP deals with Case (i), (ii) or (iii), the desired conclusion follows from the clean environment assumption. Assume now that the $i$th iteration of LOOP deals with Case (iv). Consider any leaf $x$ of $\mathit{Tree}\langle\Psi_{i+1}\rangle$. As noted when discussing Case (iv) in the proof of Clause (b), $\mathit{Tree}\langle\Psi_i\rangle = \mathit{Tree}\langle\Psi_{i+1}\rangle$, so $x$ is also a leaf of $\mathit{Tree}\langle\Psi_i\rangle$. Therefore, if $\Psi_{i+1}^{\preceq x} = \Psi_i^{\preceq x}$, the conclusion that $\Psi_{i+1}^{\preceq x}$ is prelegal follows from the induction hypothesis. Suppose now $\Psi_{i+1}^{\preceq x} \neq \Psi_i^{\preceq x}$. Note that then $\Psi_{i+1}^{\preceq x}$ looks like $\langle \Psi_i^{\preceq x}, \top y_1.\alpha, \ldots, \top y_m.\alpha\rangle$, where for each $y_j$ $(1 \leq j \leq m)$ we have $\overline{z} = x$ and $\underline{z} = y_j$ for some leaf $z$ of $T_i$. By the induction hypothesis for clause (d), each such $y_j$ is a leaf of $\mathit{Tree}\langle\Psi_i^{\preceq x}\rangle$. By the induction hypothesis for the present clause, $\Psi_i^{\preceq x}$ is prelegal. Adding to such a position the nonreplicative moves $\top y_1.\alpha, \ldots, \top y_m.\alpha$ — where the $y_j$ are leaves of $\mathit{Tree}\langle\Psi_i^{\preceq x}\rangle$ — cannot bring it to a non-prelegal position. Thus, $\Psi_{i+1}^{\preceq x}$ remains prelegal.

*Clauses (d) and (e):* If the $i$th iteration of LOOP deals with Cases (iii) or (iv), $T_i$ is not modified, and no moves of the form $x$: or $x.y$: (where $x, y$ are bit strings) are made in the consequent of $\mathbin{!}F^* \rightarrow \mathbin{!}\mathbin{!}F^*$, so $\mathit{Tree}\langle\Psi_i\rangle$ and $\mathit{Tree}\langle\Psi_i^{\preceq x}\rangle$ (any leaf $x$ of $\mathit{Tree}\langle\Psi_i\rangle$) are not affected, either. Hence Clauses (d) and (e) for $i+1$ are automatically inherited from the induction hypothesis for these clauses. This inheritance also takes place — even if no longer "automatically" — when the $i$th iteration of LOOP deals with Case (i) or (ii). This can be verified by a routine analysis of how Cases (i) and (ii) modify $T_i$ and the other relevant parameters. Details are left to the reader.

*Clause (f):* Consider any leaf $z$ of $T_{i+1}$. When the $i$th iteration of LOOP deals with Case (i) or (ii), no moves of the form $x.\alpha$ are made in the antecedent of $\mathbin{!}F^* \rightarrow \mathbin{!}\mathbin{!}F^*$, and no moves of the form $x.y.\alpha$ are made in the consequent (any bit strings $x, y$). Based on this, it is easy to see that for all bit strings $x, y$ — including the case $x = \overline{z}$ and $y = \underline{z}$ — we have $\Phi_{i+1}^{\preceq x} = \Phi_i^{\preceq x}$ and $(\Psi_{i+1}^{\preceq x})^{\preceq y} = (\Psi_i^{\preceq x})^{\preceq y}$. Hence clause (f) for $i+1$ is inherited from the same clause for $i$. Now suppose the $i$th iteration of LOOP deals with Case (iii). Then $T_{i+1} = T_i$ and hence $z$ is also a leaf of $T_i$. From the description of Case (iii) one can easily see that if $w \not\preceq \overline{z}$ or $u \not\preceq \underline{z}$, we have $\Phi_{i+1}^{\preceq\overline{z}} = \Phi_i^{\preceq\overline{z}}$ and $(\Psi_{i+1}^{\preceq\overline{z}})^{\preceq\underline{z}} = (\Psi_i^{\preceq\overline{z}})^{\preceq\underline{z}}$, so the equation $\Phi_{i+1}^{\preceq\overline{z}} = (\Psi_{i+1}^{\preceq\overline{z}})^{\preceq\underline{z}}$ is true by the induction hypothesis; and if $w \preceq \overline{z}$ and $u \preceq \underline{z}$, then $\Phi_{i+1}^{\preceq\overline{z}} = \langle \Phi_i^{\preceq\overline{z}}, \bot\alpha\rangle$ and $(\Psi_{i+1}^{\preceq\overline{z}})^{\preceq\underline{z}} = \langle (\Psi_i^{\preceq\overline{z}})^{\preceq\underline{z}}, \bot\alpha\rangle$. But, by the induction hypothesis, $\Phi_i^{\preceq\overline{z}} = (\Psi_i^{\preceq\overline{z}})^{\preceq\underline{z}}$. Hence $\Phi_{i+1}^{\preceq\overline{z}} = (\Psi_{i+1}^{\preceq\overline{z}})^{\preceq\underline{z}}$. A similar argument applies when the $i$th iteration of LOOP deals with Case (iv).

*Clause (g):* Note that all of the moves made in any of Cases (i)-(iv) of LOOP have the prefix '1.' or '2.', i.e. are made either in the antecedent or the consequent of $!!F^* \rightarrow !!!!F^*$. Hence, in order to show that $\Theta_{i+1}$ is a legal position of $!!F^* \rightarrow !!!!F^*$, it would suffice to verify that $\Phi_{i+1} \in \mathbf{Lr}^{!!F^*}$ and $\Psi_{i+1} \in \mathbf{Lr}^{!!!!F^*}$.

Suppose the $i$th iteration of LOOP deals with Case (i) or (ii). The clean environment assumption guarantees that $\Psi_{i+1} \in \mathbf{Lr}^{!!!!F^*}$. In the antecedent of $!!F^* \rightarrow !!!!F^*$ only replicative moves are made. Replicative moves can yield an illegal position ($\Phi_{i+1}$ in our case) of a $!!$-game only if they yield a non-prelegal position. But, by clause (a), $\Phi_{i+1}$ is prelegal. Hence it is a legal position of $!!F^*$.

Suppose now the $i$th iteration of LOOP deals with Case (iii). Again, that $\Psi_{i+1} \in \mathbf{Lr}^{!!!!F^*}$ is guaranteed by the clean environment assumption. So, we only need to verify that $\Phi_{i+1} \in \mathbf{Lr}^{!!F^*}$. By clause (a), this position is prelegal. So, it remains to see that, for any leaf $y$ of $\mathit{Tree}\langle\Phi_{i+1}\rangle$, $\Phi_{i+1}^{\preceq z} \in \mathbf{Lr}^{!!F^*}$. Pick an arbitrary leaf $y$ of $\mathit{Tree}\langle\Phi_{i+1}\rangle$ — i.e., by clause (a), of $\overline{T}_{i+1}$. Let $z$ be the leaf of $T_{i+1}$ with $y = \overline{z}$. We already know that $\Psi_{i+1} \in \mathbf{Lr}^{!!!!F^*}$. By clause (d), we also know that $\overline{z}$ is a leaf of $\mathit{Tree}\langle\Psi_{i+1}\rangle$. Consequently, $\Psi_{i+1}^{\preceq \overline{z}} \in \mathbf{Lr}^{!!F^*}$. Again by clause (d), $\underline{z}$ is a leaf of $\mathit{Tree}\langle\Psi_{i+1}^{\preceq \overline{z}}\rangle$. Hence, $(\Psi_{i+1}^{\preceq \overline{z}})^{\preceq \underline{z}}$ should be a legal position of $F^*$. But, by clause (f), $\Phi_{i+1}^{\preceq \overline{z}} = (\Psi_{i+1}^{\preceq \overline{z}})^{\preceq \underline{z}}$. Thus, $\Phi_{i+1}^{\preceq y} \in \mathbf{Lr}^{F^*}$.

Finally, suppose the $i$th iteration of LOOP deals with Case (iv). By the clean environment assumption, $\Phi_{i+1} \in \mathbf{Lr}^{!!F^*}$. Now consider $\Psi_{i+1}$. This position is prelegal by clause (b). So, in order for $\Psi_{i+1}$ to be a legal position of $!!!!F^*$, for every leaf $x$ of $\mathit{Tree}\langle\Psi_{i+1}\rangle$ we should have $\Psi_{i+1}^{\preceq x} \in \mathbf{Lr}^{!!F^*}$. Consider an arbitrary such leaf $x$. By clause (c), $\Psi_{i+1}^{\preceq x}$ is prelegal. Hence, a sufficient condition for $\Psi_{i+1}^{\preceq x} \in \mathbf{Lr}^{!!F^*}$ is that, for every leaf $y$ of $\mathit{Tree}\langle\Psi_{i+1}^{\preceq x}\rangle$, $(\Psi_{i+1}^{\preceq x})^{\preceq y} \in \mathbf{Lr}^{F^*}$. So, let $y$ be an arbitrary such leaf. By clause (e), there is a leaf $z$ of $T_{i+1}$ such that $\overline{z} = x$ and $\underline{z} = y$. Therefore, by clause (f), $\Phi_{i+1}^{\preceq \overline{z}} = (\Psi_{i+1}^{\preceq x})^{\preceq y}$. But we know that $\Phi_{i+1} \in \mathbf{Lr}^{!!F^*}$ and hence (with clause (a) in mind) $\Phi_{i+1}^{\preceq \overline{z}} \in \mathbf{Lr}^{F^*}$. Consequently, $(\Psi_{i+1}^{\preceq x})^{\preceq y} \in \mathbf{Lr}^{F^*}$. □

**Lemma 38.** *For every finite initial segment $\Upsilon$ of $\Theta$, there is $i \in N$ such that $\Upsilon$ is a (not necessarily proper) initial segment of $\Theta_i$ and hence of every $\Theta_j$ with $i \leq j \in N$.*

*Proof.* The statement of the lemma is straightforward when there are infinitely many iterations of LOOP, for each iteration adds a nonzero number of new moves to the run and hence there are arbitrarily long $\Theta_i$s, each of them being an initial segment of $\Theta$. Suppose now LOOP is iterated a finite number $m$ of times. It would be (necessary and) sufficient to verify that in this case $\Theta = \Theta_m$, i.e. no moves are made during the last iteration of LOOP. But this is indeed so. From the description of LOOP we see that the machine does not make any moves during a given iteration unless the environment makes a move $\gamma$ first. So, assume $\bot$ makes move $\gamma$ during the $m$th iteration of LOOP. By the clean environment assumption, we should have $\langle\Theta_m, \bot\gamma\rangle \in \mathbf{Lr}^{!!F^* \rightarrow !!!!F^*}$. It is easy to see that such a $\gamma$ would have to satisfy the conditions of one of the Cases (i)-(iv) of LOOP. But then there would

be an $(m+1)$th iteration of LOOP, contradicting out assumption that there are only $m$ iterations. □

Let us use $\Phi$ and $\Psi$ to denote $\neg\Theta^{1\cdot}$ and $\Theta^{2\cdot}$, respectively. Of course, the statement of Lemma 38 is true for $\Phi$ and $\Psi$ (instead of $\Theta$) as well. Taking into account that, by definition, a given run is legal if all of its finite initial segments are legal, the following fact is an immediate corollary of Lemmas 38 and 37(g):

$$\Theta \in \mathbf{Lr}^{\mathfrak{b}F^*\to\mathfrak{b}\mathfrak{b}F^*}. \text{ Hence, } \Phi \in \mathbf{Lr}^{\mathfrak{b}F^*} \text{ and } \Psi \in \mathbf{Lr}^{\mathfrak{b}\mathfrak{b}F^*}. \tag{6}$$

To complete our proof of Lemma 34, we need to show that $\mathbf{Wn}^{\mathfrak{b}F^*\to\mathfrak{b}\mathfrak{b}F^*}\langle\Theta\rangle = \top$. With (6) in mind, if $\mathbf{Wn}^{\mathfrak{b}\mathfrak{b}F^*}\langle\Psi\rangle = \top$, we are done. Assume now $\mathbf{Wn}^{\mathfrak{b}\mathfrak{b}F^*}\langle\Psi\rangle = \bot$. Then, by the definition of $\mathfrak{b}$, there is an infinite bit string $x$ such that $\Psi^{\preceq x}$ is a legal but lost (by $\top$) run of $\mathfrak{b}F^*$. This means that, for some infinite bit string $y$,

$$\mathbf{Wn}^{F^*}\langle(\Psi^{\preceq x})^{\preceq y}\rangle = \bot. \tag{7}$$

Fix these $x$ and $y$. For each $i \in N$, let $x_i$ denote the (obviously unique) leaf of $\mathit{Tree}\langle\Psi_i\rangle$ such that $x_i \preceq x$; and let $y_i$ denote the (again unique) leaf of $\mathit{Tree}\langle\Psi_i^{\preceq x_i}\rangle$ such that $y_i \preceq y$. Next, let $z_i$ denote the leaf of $T_i$ with $\overline{z}_i = x_i$ and $\underline{z}_i = y_i$. According to Lemma 37(e), such a $z_i$ exists and is unique.

Consider any $i$ with $i+1 \in N$. Clearly $x_i \preceq x_{i+1}$ and $y_i \preceq y_{i+1}$. By our choice of the $z_j$, we then have $\overline{z}_i \preceq \overline{z}_{i+1}$ and $\underline{z}_i \preceq \underline{z}_{i+1}$. Hence, by Lemma 36, $z_i \preceq z_{i+1}$. Let us fix an infinite bit string $z$ such that for every $i \in N$, $\overline{z}_i \preceq z$. Based on the just-made observation that we always have $z_i \preceq z_{i+1}$, such a $z$ exists.

In view of Lemma 38, Lemma 37(f) easily allows us to find that $\Phi^{\preceq z} = (\Psi^{\preceq x})^{\preceq y}$. Therefore, by (7), $\mathbf{Wn}^{F^*}\langle\Phi^{\preceq z}\rangle = \bot$. By the definition of $\mathfrak{b}$, this means that $\mathbf{Wn}^{\mathfrak{b}F^*}\langle\Phi\rangle = \bot$. Hence, by the definition of $\to$ and with (6) in mind, $\mathbf{Wn}^{\mathfrak{b}F^*\to\mathfrak{b}\mathfrak{b}F^*}\langle\Theta\rangle = \top$. Done. □

# 12 Proof of Theorem 12

Now we are ready to prove our main Theorem 12. Consider an arbitrary sequent $S$ with $\mathbf{INT} \vdash S$. By induction on the **INT**-derivation of $S$, we are going to show that $S$ has a uniform solution $\mathcal{E}$. This is sufficient to conclude that **INT** is 'uniformly sound'. The theorem also claims 'constructive soundness', i.e. that such an $\mathcal{E}$ can be effectively built from a given **INT**-derivation of $S$. This claim of the theorem will be automatically taken care of by the fact that our proof of the existence of $\mathcal{E}$ is constructive: the uniform-validity and closure lemmas on which we rely provide a way for actually constructing a corresponding uniform solution. With this remark in mind and for the considerations of readability, in what follows we only talk about uniform validity without explicitly mentioning uniform solutions for the corresponding formulas/sequents and without explicitly showing how to construct such solutions. Also, we no longer use $\Rightarrow$ or $\circ\!\!-$, seeing each sequent $\underline{F} \Rightarrow K$ as the formula $\underline{\mathfrak{b}F} \to K$ and each subformula $E_1 \circ\!\!- E_2$ of such a formula as $\mathfrak{b}E_1 \to E_2$.

This is perfectly legitimate because, by definition, $(\underline{F} \Rightarrow K)^* = (\underline{\circ}F \rightarrow K)^*$ and $(E_1 \circ\!\!- E_2)^* = (\circ E_1 \rightarrow E_2)^*$.

There are 15 cases to consider, corresponding to the 15 possible rules that might have been used at the last step of an **INT**-derivation of $S$, with $S$ being the conclusion of the rule. In each non-axiom case below, "induction hypothesis" means the assumption that the premise(s) of the corresponding rule is (are) uniformly valid. The goal in each case is to show that the conclusion of the rule is also uniformly valid. "Modus ponens" should be understood as Lemma 21, and "transitivity" as Lemma 22.

**Identity:** Immediately from Lemma 23.

**Domination:** Immediately from Lemma 27.

**Exchange:** By the induction hypothesis, $\Vdash \underline{\circ}G \wedge \circ E \wedge \circ F \wedge \underline{\circ}H \rightarrow K$. And, by Lemma 17(a), $\Vdash (\underline{\circ}G \wedge \circ E \wedge \circ F \wedge \underline{\circ}H \rightarrow K) \rightarrow (\underline{\circ}G \wedge \circ F \wedge \circ E \wedge \underline{\circ}H \rightarrow K)$. Applying modus ponens yields $\Vdash \underline{\circ}G \wedge \circ F \wedge \circ E \wedge \underline{\circ}H \rightarrow K$.

**Weakening:** Similar to the previous case, using Lemma 17(b) instead of 17(a).

**Contraction**: By Lemma 17(c) (with empty $\underline{U}$), $\Vdash (\circ F \rightarrow \circ F \wedge \circ F) \rightarrow (\underline{\circ}G \wedge \circ F \rightarrow \underline{\circ}G \wedge \circ F \wedge \circ F)$. And, by Lemma 26, $\Vdash \circ F \rightarrow \circ F \wedge \circ F$. Hence, by modus ponens, $\Vdash \underline{\circ}G \wedge \circ F \rightarrow \underline{\circ}G \wedge \circ F \wedge \circ F$. But, by the induction hypothesis, $\Vdash \underline{\circ}G \wedge \circ F \wedge \circ F \rightarrow K$. Hence, by transitivity, $\Vdash \underline{\circ}G \wedge \circ F \rightarrow K$.

**Right** $\circ\!\!-$: From Lemma 17(d), $\Vdash (\underline{\circ}G \wedge \circ F \rightarrow K) \rightarrow (\underline{\circ}G \rightarrow (\circ F \rightarrow K))$. And, by the induction hypothesis, $\Vdash \underline{\circ}G \wedge \circ F \rightarrow K$. Applying modus ponens, we get $\Vdash \underline{\circ}G \rightarrow (\circ F \rightarrow K)$.

**Left** $\circ\!\!-$: By the induction hypothesis,

$$\Vdash \underline{\circ}G \wedge \circ F \rightarrow K_1; \tag{8}$$

$$\Vdash \underline{\circ}H \rightarrow K_2. \tag{9}$$

Our goal is to show that

$$\Vdash \underline{\circ}G \wedge \underline{\circ}H \wedge \circ(\circ K_2 \rightarrow F) \rightarrow K_1. \tag{10}$$

By Lemma 18, (9) implies $\Vdash \circ(\underline{\circ}H \rightarrow K_2)$. Also, by Lemma 24, $\Vdash \circ(\underline{\circ}H \rightarrow K_2) \rightarrow (\circ\underline{\circ}H \rightarrow \circ K_2)$. Applying modus ponens, we get $\Vdash \circ\underline{\circ}H \rightarrow \circ K_2$. Again using Lemma 18, we find $\Vdash \circ(\circ\underline{\circ}H \rightarrow \circ K_2)$, which, (again) by Lemma 24 and modus ponens, implies

$$\Vdash \circ\circ\underline{\circ}H \rightarrow \circ\circ K_2. \tag{11}$$

Combining Lemmas 17(c) (with empty $\underline{W}, \underline{U}$) and 34, by modus ponens, we find $\Vdash \underline{\circ}H \rightarrow \underline{\circ\circ}H$. Next, by lemma 25, $\Vdash \underline{\circ\circ}H \rightarrow \circ\underline{\circ}H$. Hence, by transitivity, $\Vdash \underline{\circ}H \rightarrow \circ\underline{\circ}H$. At the same time, by Lemma 34, $\Vdash \circ\underline{\circ}H \rightarrow \circ\circ\underline{\circ}H$. Again by

transitivity, $\Vdash \underline{\mathsf{o}}H \to \mathsf{o}\mathsf{o}\underline{\mathsf{o}}H$. This, together with (11), by transitivity, yields

$$\Vdash \underline{\mathsf{o}}H \to \mathsf{o}\mathsf{o}K_2. \tag{12}$$

Next, by Lemma 24,

$$\Vdash \mathsf{o}(\mathsf{o}K_2 \to F) \to (\mathsf{o}\mathsf{o}K_2 \to \mathsf{o}F). \tag{13}$$

From Lemma 17(e), $\Vdash (\mathsf{o}(\mathsf{o}K_2 \to F) \to (\mathsf{o}\mathsf{o}K_2 \to \mathsf{o}F)) \wedge (\underline{\mathsf{o}}H \to \mathsf{o}\mathsf{o}K_2) \to (\mathsf{o}(\mathsf{o}K_2 \to F) \to (\underline{\mathsf{o}}H \to \mathsf{o}F))$. This, together with (13) and (12), by modus ponens, yields

$$\Vdash \mathsf{o}(\mathsf{o}K_2 \to F) \to (\underline{\mathsf{o}}H \to \mathsf{o}F). \tag{14}$$

By Lemma 17(f),

$$\begin{array}{l} \Vdash (\mathsf{o}(\mathsf{o}K_2 \to F) \to (\underline{\mathsf{o}}H \to \mathsf{o}F)) \wedge (\underline{\mathsf{o}}G \wedge \mathsf{o}F \to K_1) \\ \to (\underline{\mathsf{o}}G \wedge \underline{\mathsf{o}}H \wedge \mathsf{o}(\mathsf{o}K_2 \to F) \to K_1). \end{array} \tag{15}$$

From (14), (8) and (15), by modus ponens, we obtain the desired (10).

**Right $\sqcap$:** By the induction hypothesis, $\Vdash \underline{\mathsf{o}}G \to K_1, \ldots, \Vdash \underline{\mathsf{o}}G \to K_n$. And, from Lemma 17(h), $\Vdash (\underline{\mathsf{o}}G \to K_1) \wedge \ldots \wedge (\underline{\mathsf{o}}G \to K_n) \to (\underline{\mathsf{o}}G \to K_1 \sqcap \ldots \sqcap K_n)$. Modus ponens yields $\Vdash \underline{\mathsf{o}}G \to K_1 \sqcap \ldots \sqcap K_n$.

**Left $\sqcap$:** By Lemma 28(a), $\Vdash \mathsf{o}(F_1 \sqcap \ldots \sqcap F_n) \to \mathsf{o}F_i$; and, by Lemma 17(c), $\Vdash (\mathsf{o}(F_1 \sqcap \ldots \sqcap F_n) \to \mathsf{o}F_i) \to (\underline{\mathsf{o}}G \wedge \mathsf{o}(F_1 \sqcap \ldots \sqcap F_n) \to \underline{\mathsf{o}}G \wedge \mathsf{o}F_i)$. Modus ponens yields $\underline{\mathsf{o}}G \wedge \mathsf{o}(F_1 \sqcap \ldots \sqcap F_n) \to \underline{\mathsf{o}}G \wedge \mathsf{o}F_i$. But, by the induction hypothesis, $\Vdash \underline{\mathsf{o}}G \wedge \mathsf{o}F_i \to K$. So, by transitivity, $\Vdash \underline{\mathsf{o}}G \wedge \mathsf{o}(F_1 \sqcap \ldots \sqcap F_n) \to K$.

**Right $\sqcup$:** By the induction hypothesis, $\Vdash \underline{\mathsf{o}}G \to K_i$. According to Lemma 17(j), $\Vdash (\underline{\mathsf{o}}G \to K_i) \to (\underline{\mathsf{o}}G \to K_1 \sqcup \ldots \sqcup K_n)$. Therefore, by modus ponens, $\Vdash \underline{\mathsf{o}}G \to K_1 \sqcup \ldots \sqcup K_n$.

**Left $\sqcup$:** By the induction hypothesis, $\Vdash \underline{\mathsf{o}}G \wedge \mathsf{o}F_1 \to K, \ldots, \Vdash \underline{\mathsf{o}}G \wedge \mathsf{o}F_n \to K$. And, by Lemma 17(i), $\Vdash (\underline{\mathsf{o}}G \wedge \mathsf{o}F_1 \to K) \wedge \ldots \wedge (\underline{\mathsf{o}}G \wedge \mathsf{o}F_n \to K) \to (\underline{\mathsf{o}}G \wedge (\mathsf{o}F_1 \sqcup \ldots \sqcup \mathsf{o}F_n) \to K)$. Hence, by modus ponens,

$$\Vdash \underline{\mathsf{o}}G \wedge (\mathsf{o}F_1 \sqcup \ldots \sqcup \mathsf{o}F_n) \to K. \tag{16}$$

Next, by Lemma 17(c), $\Vdash (\mathsf{o}(F_1 \sqcup \ldots \sqcup F_n) \to \mathsf{o}F_1 \sqcup \ldots \sqcup \mathsf{o}F_n) \to (\underline{\mathsf{o}}G \wedge \mathsf{o}(F_1 \sqcup \ldots \sqcup F_n) \to \underline{\mathsf{o}}G \wedge (\mathsf{o}F_1 \sqcup \ldots \sqcup \mathsf{o}F_n))$. But, by Lemma 28(c), $\Vdash \mathsf{o}(F_1 \sqcup \ldots \sqcup F_n) \to \mathsf{o}F_1 \sqcup \ldots \sqcup \mathsf{o}F_n$. Modus ponens yields $\Vdash \underline{\mathsf{o}}G \wedge \mathsf{o}(F_1 \sqcup \ldots \sqcup F_n) \to \underline{\mathsf{o}}G \wedge (\mathsf{o}F_1 \sqcup \ldots \sqcup \mathsf{o}F_n)$. From here and (16), by transitivity, $\Vdash \underline{\mathsf{o}}G \wedge \mathsf{o}(F_1 \sqcup \ldots \sqcup F_n) \to K$.

**Right $\bigsqcap$:** First, consider the case when $\underline{\mathsf{o}}G$ is nonempty. By the induction hypothesis, $\Vdash \underline{\mathsf{o}}G \to K(y)$. Therefore, by Lemma 19, $\Vdash \bigsqcap y(\underline{\mathsf{o}}G \to K(y))$ and, by Lemma 29 and modus ponens, $\Vdash \bigsqcap y \underline{\mathsf{o}}G \to \bigsqcap y K(y)$. At the same time, by Lemma

32, $\vdash \underline{\downarrow G} \rightarrow \sqcap y \underline{\downarrow G}$. By transitivity, we then get $\vdash \underline{\downarrow G} \rightarrow \sqcap y K(y)$. But, by Lemma 33, $\vdash \sqcap y K(y) \rightarrow \sqcap x K(x)$. Transitivity yields $\vdash \underline{\downarrow G} \rightarrow \sqcap x K(x)$. The case when $\underline{\downarrow G}$ is empty is simpler, for then $\vdash \underline{\downarrow G} \rightarrow \sqcap x K(x)$, i.e. $\vdash \sqcap x K(x)$, can be obtained directly from the induction hypothesis by Lemmas 19, 33 and modus ponens.

**Left** $\sqcap$: Similar to Left $\sqcap$, only using Lemma 28(b) instead of 28(a).

**Right** $\sqcup$: By the induction hypothesis, $\vdash \underline{\downarrow G} \rightarrow K(t)$. And, by Lemma 31, $\vdash K(t) \rightarrow \sqcup x K(x)$. Transitivity yields $\vdash \underline{\downarrow G} \rightarrow \sqcup x K(x)$.

**Left** $\sqcup$: By the induction hypothesis, $\vdash \underline{\downarrow G} \wedge \downarrow F(y) \rightarrow K$. This, by Lemma 19, implies $\vdash \sqcap y (\underline{\downarrow G} \wedge \downarrow F(y) \rightarrow K)$. From here, by Lemma 30 and modus ponens, we get

$$\vdash \sqcap y \underline{\downarrow G} \wedge \sqcup y \downarrow F(y) \rightarrow \sqcup y K. \tag{17}$$

By Lemma 17(c), $\vdash (\underline{\downarrow G} \rightarrow \sqcap y \underline{\downarrow G}) \rightarrow (\underline{\downarrow G} \wedge \sqcup y \downarrow F(y) \rightarrow \sqcap y \underline{\downarrow G} \wedge \sqcup y \downarrow F(y))$. This, together with Lemma 32, by modus ponens, implies $\underline{\downarrow G} \wedge \sqcup y \downarrow F(y) \rightarrow \sqcap y \underline{\downarrow G} \wedge \sqcup y \downarrow F(y)$. From here and (17), by transitivity, $\vdash \underline{\downarrow G} \wedge \sqcup y \downarrow F(y) \rightarrow K$. But, by Lemmas 33, 17(c) and modus ponens, $\vdash \underline{\downarrow G} \wedge \sqcup x \downarrow F(x) \rightarrow \underline{\downarrow G} \wedge \sqcup y \downarrow F(y)$. Hence, by transitivity,

$$\vdash \underline{\downarrow G} \wedge \sqcup x \downarrow F(x) \rightarrow K. \tag{18}$$

Next, by Lemma 17(c), $\vdash (\downarrow \sqcup x F(x) \rightarrow \sqcup x \downarrow F(x)) \rightarrow (\underline{\downarrow G} \wedge \downarrow \sqcup x F(x) \rightarrow \underline{\downarrow G} \wedge \sqcup x \downarrow F(x))$. But, by Lemma 28(d), $\vdash \downarrow \sqcup x F(x) \rightarrow \sqcup x \downarrow F(x)$. Modus ponens yields $\underline{\downarrow G} \wedge \downarrow \sqcup x F(x) \rightarrow \underline{\downarrow G} \wedge \sqcup x \downarrow F(x)$. From here and (18), by transitivity, $\vdash \underline{\downarrow G} \wedge \downarrow \sqcup x F(x) \rightarrow K$. □

# References


[1] Blass, A. Degrees of indeterminacy of games. *Fundamenta Mathematicae*, 77:151–166, 1972.

[2] Blass, A. A game semantics for linear logic. *Annals of Pure and Applied Logic*, 56:183–220, 1992.

[3] Felscher, W. Dialogues, strategies, and intuitionistic provability. *Annals of Pure and Applied Logic*, 28: 217–254, 1885.

[4] J.Y. Girard, J. Y. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[5] Gödel, K. Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes. *Dialectica*, 12:280–287, 1958.

[6] Japaridze, G. A constructive game semantics for the language of linear logic. *Annals of Pure and Applied Logic*, 85:87–156, 1997.

[7] Japaridze, G. Introduction to computability logic. *Annals of Pure and Applied Logic*, 123:1–99, 2003.

[8] Japaridze, G. From truth to computability I. *Theoretical Computer Science*, 357:100–135, 2006.

[9] Japaridze, G. From truth to computability II. *Theoretical Computer Science*, 376:20–52, 2007.

[10] Japaridze, G. Computability logic: a formal theory of interaction. In Goldin, Dina, Smolka, Scott and Wegner, Peter, editors, *Interactive Computation: The New Paradigm*, pages 183–223. Springer, 2006.

[11] Japaridze, G. Propositional computability logic I. *ACM Transactions on Computational Logic*, 7:302–330, 2006.

[12] Japaridze, G. Propositional computability logic II. *ACM Transactions on Computational Logic*, 7:331–362, 2006.

[13] Kleene, S. C. *Introduction to Metamathematics.* D. van Nostrand Company, New York, Toronto, 1952.

[14] Kolmogorov, A. N. Zur Deutung der intuitionistischen Logik. *Mathematische Zeitschrift*, 35:58–65, 1932.

[15] Lorenzen, P. Ein dialogisches Konstruktivitätskriterium. In *Infinitistic Methods* (Proc. Symp. Foundations of Mathematics), pages 193–200, Warsaw, 1961. PWN.

[16] Medvedev, Y. Interpretation of logical formulas by means of finite problems and its relation to the realizability theory. *Soviet Mathematics Doklady*, 4:180-183, 1963.

# Regular Papers

Acta Cybernetica 18 (2007) 117–134.

# On monotone languages and their characterization by regular expressions

György Gyurica*

*To the memory of Balázs Imreh*

**Abstract**

In one of their papers, F. Gécseg and B. Imreh gave a characterization for monotone string languages by regular expressions. It has turned out that the monotone string languages are exactly those languages that can be represented by finite unions of seminormal chain languages. In this paper a similar characterization is given for monotone DR-languages.

## 1 Introduction

Monotone string and tree languages were introduced by Gécseg and Imreh in [4] where these languages were characterized by means of syntactic monoids. They also used chain languages to represent monotone string languages by regular expressions, and showed that any monotone string language can be represented as the union of finitely many seminormal chain languages and that, conversely, any seminormal chain language can be recognized by a monotone recognizer.

In this paper we continue the investigation of monotone string and DR-languages. Our primary goal was to characterize the monotone DR-languages by regular $\Sigma X$-expressions, but we have also introduced the concept of iterational height for regular expressions which was useful to state conditions under which iteration preserves monotonicity. The same result was adapted to DR-languages, too.

Thereafter, a simple characterization of monotone DR-languages was given. The number of the auxiliary variables used in this representation and some decomposition problems were also investigated. Later, we stated some conditions that are required to preserve monotonicity when using the operations of $x$-product and $x$-iteration. Finally, we introduced the concept of generalized $R$-chain languages, for which it will turn out that they represent exactly the monotone DR-languages. For notions and notation not defined in this paper we refer the reader to [4] and [7].

*Department of Informatics, University of Szeged, Árpád tér 2, H-6720 Szeged, Hungary. E-mail: gyurica@inf.u-szeged.hu

## 2 Monotone string languages

Let $X$ be an alphabet. The set of all words over $X$ is denoted by $X^*$. Let us denote the *length* of a word $u \in X^*$ by $|u|$ which is the number of occurrences of letters from $X$ in $u$. The empty word is denoted by $e$. The set of words with length greater than 0 is denoted by $X^+(= X^* \setminus \{e\})$.

An *$X$-recognizer* is a system $\mathbf{A} = (A, X, \delta, a_0, A')$, where $A$ is a finite set of *states*, $X$ is the *input alphabet*, $\delta : A \times X \to A$ is the *next-state function*, $a_0 \in A$ is the *initial state*, and $A' \subseteq A$ is the set of *final states*. The next-state function can be extended to a function $\delta^* : A \times X^* \to A$, where $\delta^*(a, e) = a$ and $\delta^*(a, xu) = \delta^*(\delta(a, x), u)$ $(a \in A,\ x \in X,\ u \in X^*)$. If there is no danger of confusion, instead of $\delta^*(a, u)$ we can use the notation $\delta(a, u)$ or simply $au$.

The *language* $L(\mathbf{A})$ *recognized by* $\mathbf{A}$ is given by

$$L(\mathbf{A}) = \{u \in X^* \mid a_0 u \in A'\}.$$

A language $L \subseteq X^*$ is called *regular* or *recognizable* if it can be recognized by an $X$-recognizer.

An $X$-recognizer $\mathbf{A} = (A, X, \delta, a_0, A')$ is *monotone* if there is a partial ordering $\leq$ on $A$ such that for all $a \in A$ and $x \in X$, $a \leq \delta(a, x)$ holds. It is obvious that for all $a \in A$ and $u \in X^*$, $a \leq au$ holds, too. A language $L \subseteq X^*$ is *monotone* if $L = L(\mathbf{A})$ for a monotone $X$-recognizer $\mathbf{A}$. Later we will use the fact that every partial ordering on a finite set can be extended to a linear ordering. For more details we refer the reader to [4].

A language $L \subseteq X^*$ is *fundamental*, if $L = Y^*$ for a $Y \subseteq X$. A language $L \subseteq X^*$ is a *chain language* if $L$ can be given in the form $L = L_0 x_1 L_1 x_2 \dots x_{k-1} L_{k-1} x_k L_k$, where $x_1, \dots, x_k \in X$ and every $L_i$ $(0 \leq i \leq k)$ is a product of fundamental languages. A chain language $L = L_0 x_1 L_1 x_2 \dots\dots x_{k-1} L_{k-1} x_k L_k$ is called *seminormal* if $x_i \notin L_{i-1}$ for every $1 \leq i \leq k$. $L$ is *normal* if $x_i \notin L_{i-1}$ and $x_i \notin L_i$ $(1 \leq i \leq k)$. A seminormal chain language $L = L_0 x_1 L_1 x_2 \dots x_{k-1} L_{k-1} x_k L_k$ is called *simple* if each $L_i$ $(0 \leq i \leq k)$ is fundamental.

Now we recall the main result from the corresponding section in [4].

**Theorem 1.** *A language is monotone iff it can be given as a union of finitely many seminormal chain languages.* □

Let $X$ be an alphabet. The set $RE$ of all regular expressions and the language $L(\eta)$ represented by $\eta \in RE$ are defined in parallel as follows:

- $\emptyset \in RE$, $L(\emptyset) = \emptyset$,
- $\forall x \in X : x \in RE$, $L(x) = \{x\}$,
- if $\eta_1, \eta_2 \in RE$, then $(\eta_1) + (\eta_2) \in RE$, $L((\eta_1) + (\eta_2)) = L(\eta_1) \cup L(\eta_2)$,
- if $\eta_1, \eta_2 \in RE$, then $(\eta_1)(\eta_2) \in RE$, $L((\eta_1)(\eta_2)) = L(\eta_1)L(\eta_2)$,
- if $\eta \in RE$, then $(\eta)^* \in RE$, $L((\eta)^*) = L(\eta)^*$.

Some parentheses can be omitted from regular expressions, if a precedence relation is assumed between the operations of iteration, concatenation, and union in the given order.

A regular expression $\zeta$ is called a *subexpression* of $\eta$ if $\zeta$ occurs in the inductive definition of $\eta$. The set of all subexpressions of $\eta$ will be denoted by *Sub(η)*. The operation *omission* on regular expressions is defined as follows: Let us consider $\eta_1, \eta_2 \in RE$ and the regular expressions $(\eta_1)+(\eta_2)$, $(\eta_1)(\eta_2)$ and $(\eta_1)^*$. By omitting $\eta_1$ from them we get $\eta_2$ from the first two ones, and the expression $\eta_1$ from the third one. We also allow the omission of $\eta_1$ from $(\eta_1)^*$ to result in $(\emptyset)^*$. If we omit $\eta_2$ from $(\eta_1)+(\eta_2)$ and $(\eta_1)(\eta_2)$ we get $\eta_1$ and $\eta_1$ respectively. This way, omission is not well-defined, nor does it have to be. Let $\zeta$ be a subexpression of a regular expression $\eta$. We call $\zeta$ *redundant in* $\eta$ if $\zeta$ can be omitted from $\eta$ so that $L(\eta)$ remains the same after the omission. A regular expression is *reduced* if it has no redundant subexpressions.

The reduction of a regular expression is not necessarily unique as the following example shows.

**Example 2.** Let us consider the regular expression $\eta = x(yx)^* + z + (xy)^*x$. Obviously the first and the third member of the union represent the same language, that is, both of them are redundant in $\eta$. If we omit one of them separately, we get two different reduced regular expressions: $x(yx)^* + z$ and $z + (xy)^*x$ which represent the same language.

Now we define the concept of *iterational height* which is used to identify the length of the longest word that will be used in the iteration of a particular language. Let $\eta$ be a reduced regular expression in form $(\zeta)^*$. The nonnegative integer $max\{|u| : u \in L(\zeta)\}$ will be called the *iterational height* of $\eta$ (or $ih(\eta)$ for short), if $L(\zeta)$ is finite. If $L(\zeta)$ is infinite, then let $ih(\eta)$ be the infinity $\infty$ that we will treat as the largest integer. Let now $\eta$ be a reduced regular expression in any form. We define $ih(\eta)$ as $max\{ih((\zeta)^*) \mid (\zeta)^* \in Sub(\eta)\}$, if $Sub(\eta)$ contains an expression in form $(\zeta)^*$, and 0 otherwise. The iterational height of a regular language $L$ (or $ih(L)$ for short) is defined as $min\{ih(\eta) \mid L = L(\eta), \eta \in RE\}$.

**Example 3.** Let us take the regular expression $\zeta = xx + xxx$. By the definition of $ih((\zeta)^*)$ we have $ih((\zeta)^*) = 3$. Let us now consider the regular expression $\eta = x + (\zeta)^*$. It is easy to see that $ih(\eta) = 3$, because $\eta$ has a subexpression in form $(\zeta)^*$, for which $ih((\zeta)^*) = 3$. Let us now take the language $L(\eta)$, for which we get that $ih(L(\eta)) = 1$, because $L(\eta)$ can also be represented by the regular expression $(x)^*$, for which $ih((x)^*) = 1$.

**Lemma 4.** *Let $\eta$ be a reduced regular expression of the form $(\zeta)^*$. If $L(\eta)$ is monotone, then $ih(L(\eta)) \leq 1$.*

*Proof.* Let $\eta$ be a reduced regular expression of the form $(\zeta)^*$, and let the monotone $X$-recognizer $\mathbf{A}$ recognize $L(\eta)$ with the partial ordering $\leq$ on $A$. We can suppose without the loss of generality that $\mathbf{A}$ is reduced and connected from its initial state, hence there is exactly one final state $a \in A'$ such that $au = a$ for every $u \in L(\zeta)$.

Using the monotonicity of $\mathbf{A}$ we get that $ax = a$ holds for any letter $x$ from the words of $L(\zeta)$. We see that there is no such state $a' \in A \setminus \{a\}$ for which $a' \leq a$ and $a'x = a'$ hold for any $x \in X$, and we also see that there is no final state $a'' \neq a$ such that $a \leq a''$. Hence $\eta$ can be written in form $\zeta'\zeta''$, where $\zeta'$ does not contain the operation $^*$, and represents the set of all words taking $\mathbf{A}$ from $a_0$ to $a$, and where $\zeta''$ is in form $(y_1 + \ldots + y_r)^*$, where $y_1, \ldots, y_r$ are the letters from the words of $L(\zeta)$. Since $L(\eta) = L(\zeta'\zeta'')$ and $ih(L(\zeta'\zeta'')) = 1$, we get that $ih(L(\eta)) \leq 1$. □

# 3 Monotone DR-languages

A *ranked alphabet* is a finite nonempty set of operational symbols, which will be denoted by $\Sigma$. The subset of all $m$-ary operational symbols of $\Sigma$ will be denoted by $\Sigma_m$. We shall suppose in the rest of this paper that $\Sigma_0 = \emptyset$. Let $\mathfrak{p}(S)$ stand for the power set of the set $S$.

Let $X$ be a set of variables. The set $T_\Sigma(X)$ of *$\Sigma X$-trees* is defined as follows:

(i) $X \subseteq T_\Sigma(X)$,

(ii) $\sigma(p_1, \ldots, p_m) \in T_\Sigma(X)$, if $m \geq 0$, $\sigma \in \Sigma_m$ and $p_1, \ldots, p_m \in T_\Sigma(X)$,

(iii) every $\Sigma X$-tree can be obtained by applying the rules $(i)$ and $(ii)$ a finite number of times.

In the rest of this paper $X$ will stand for the countable set $\{x_1, x_2, \ldots\}$, and for every $n \geq 0$, $X_n$ will denote the subset $\{x_1, \ldots, x_n\}$ of $X$.

A pair $\mathcal{A} = (A, \Sigma)$ will represent a *deterministic root-to-frontier $\Sigma$-algebra* (or a DR $\Sigma$-*algebra* for short), where $A$ is a nonempty set, and $\Sigma$ is a ranked alphabet. Every $\sigma \in \Sigma_m$ is represented as a mapping $\sigma^{\mathcal{A}} : A \to A^m$. $\mathcal{A}$ is called *finite*, if $\Sigma$ is a ranked alphabet and $A$ is finite.

A *deterministic root-to-frontier $\Sigma X_n$-recognizer* (or a DR $\Sigma X_n$-*recognizer* for short) is a system $\mathfrak{A} = (\mathcal{A}, a_0, \mathbf{a})$, where $\mathcal{A} = (A, \Sigma)$ is a finite DR $\Sigma$-algebra, $a_0 \in A$ is the *initial state*, and $\mathbf{a} = (A^{(1)}, \ldots, A^{(n)}) \in \mathfrak{p}(A)^n$ is the *final state vector*. If $\Sigma$ or $X_n$ is not specified, we speak of *DR-recognizers*.

Let $\mathfrak{A} = (\mathcal{A}, a_0, \mathbf{a})$ be a DR $\Sigma X_n$-recognizer, and let the mapping $\alpha : T_\Sigma(X_n) \to \mathfrak{p}(A)$ be defined as follows. For every $p \in T_\Sigma(X_n)$

(i) if $p = x_i \in X_n$, then $\alpha(p) = A^{(i)}$,

(ii) if $p = \sigma(p_1, \ldots, p_m)$, then $\alpha(p) = \{a \in A \mid \sigma^{\mathcal{A}}(a) \in \alpha(p_1) \times \ldots \times \alpha(p_m)\}$.

The *tree language $T(\mathfrak{A})$ recognized* by $\mathfrak{A}$ can be given by

$$T(\mathfrak{A}) = \{p \in T_\Sigma(X_n) \mid a_0 \in \alpha(p)\}.$$

Tree languages that can be recognized by DR-recognizers are also called *DR-languages*.

Let $\mathfrak{A}$ be a DR $\Sigma X_n$-recognizer and $a \in A$ one of its states. We define the tree language $T(\mathfrak{A}, a)$ as the set $\{ p \in T_\Sigma(X_n) \mid a \in \alpha(p)\}$. A state $a$ is called

0-*state* if $T(\mathfrak{A}, a) = \emptyset$. $\mathfrak{A}$ is called *normalized* if for all $\sigma \in \Sigma_m$ and $a \in A$ it holds that each component of $\sigma^{\mathcal{A}}(a)$ is a 0-state or no component of $\sigma^{\mathcal{A}}(a)$ is a 0-state. Moreover, $\mathfrak{A}$ is called *reduced* if for any states $a, b \in A$ it holds that $a \neq b$ implies $T(\mathfrak{A}, a) \neq T(\mathfrak{A}, b)$. It is a well-known fact that every DR-language can be recognized by a normalized and reduced DR-recognizer (cf. [5], [6] and [7]).

Let $\pi_i$ be the $i$-th projection. A DR $\Sigma$-algebra $\mathcal{A} = (A, \Sigma)$ is called *monotone* if there is a partial ordering $\leq$ on $A$ such that $a \leq \pi_i(\sigma^{\mathcal{A}}(a))$ holds for all $a \in A$, $\sigma \in \Sigma_m$ and $1 \leq i \leq m$. We say that $\mathfrak{A}$ is a *monotone DR $\Sigma X_n$-recognizer* if the underlying DR $\Sigma$-algebra $\mathcal{A}$ is monotone. Moreover, $T \subseteq T_\Sigma(X_n)$ is a *monotone DR-language*, if $T = T(\mathfrak{A})$ for a monotone DR $\Sigma X_n$-recognizer $\mathfrak{A}$ (see, [2] and [4]).

As every partial ordering on a finite set can be extended to a linear ordering, the following lemma hold.

**Lemma 5.** *For any monotone DR-recognizer $\mathfrak{A}$ we may assume that the partial ordering on $A$ is total.*

The following lemma is also obvious.

**Lemma 6.** *Every finite DR-language is monotone.*

# 4 Basic observations

Before we continue the investigation of monotone DR-languages, we need to introduce some concepts and notions (mainly taken from [4], [6] and [7]).

Let $\Sigma$ be a ranked alphabet, and let $\hat{\Sigma}$ be an ordinary alphabet defined as follows. For all $\sigma, \tau \in \Sigma$ let

(i) $\hat{\Sigma}_\sigma = \{\sigma_1, \ldots, \sigma_m\}$, if $\sigma \in \Sigma_m$ $(m \geq 1)$, and

(ii) $\hat{\Sigma}_\sigma \cap \hat{\Sigma}_\tau = \emptyset$, if $\sigma \neq \tau$.

We define $\hat{\Sigma}$ as $\hat{\Sigma} = \bigcup(\hat{\Sigma}_\sigma \mid \sigma \in \Sigma)$. We say that the alphabet $\hat{\Sigma}$ corresponds to the ranked alphabet $\Sigma$.

Let $n \geq 1$ be fixed arbitrarily. The set $g_{x_i}(t)$ of *$x_i$-paths* of a tree $t \in T_\Sigma(X_n)$ is defined for each $i \in \{1, \ldots, n\}$ in the following way:

(i) $g_{x_i}(x_i) = \{e\}$, and $g_{x_i}(x_j) = \emptyset$, if $i \neq j$, $i, j \in \{1, \ldots, n\}$,

(ii) If $t = \sigma(t_1, \ldots, t_m)$ $(\sigma \in \Sigma_m)$, then $g_{x_i}(t) = \sigma_1 g_{x_i}(t_1) \cup \ldots \cup \sigma_m g_{x_i}(t_m)$.

For a tree language $T \subseteq T_\Sigma(X_n)$, let $g_{x_i}(T) = \bigcup_{t \in T} g_{x_i}(t)$, which is also denoted by $T_{x_i}$ $(1 \leq i \leq n)$.

Let $\Sigma$ be a ranked alphabet, and let $\hat{\Sigma}$ be the alphabet corresponding to it. Let $\mathcal{A} = (A, \Sigma)$ be a DR $\Sigma$-algebra. For every $u \in \hat{\Sigma}^*$ the mapping $u^{\mathcal{A}} : A \to A$ is defined as follows:

(i) If $u = e$, then $au^{\mathcal{A}} = a$, and

(ii) if $u = \sigma_j v$, then $au^{\mathcal{A}} = \pi_j(\sigma(a))v^{\mathcal{A}}$, $(a \in A,\ \sigma \in \Sigma_m,\ 1 \leq j \leq m,\ v \in \hat{\Sigma}^*)$.

The mapping defined above can be extended to subsets of $\hat{\Sigma}^*$ in a natural way. In the rest of this paper we will omit the superscript $\mathcal{A}$ in $u^{\mathcal{A}}$ if the DR $\Sigma$-algebra $\mathcal{A}$ inducing $u^{\mathcal{A}}$ is obvious.

A tree language $T \subseteq T_\Sigma(X_n)$ is *closed* if a tree $t \in T_\Sigma(X_n)$ is in $T$ if and only if $g_x(t) \subseteq T_x$ for all $x \in X_n$. It is a well known result, that a regular tree language is DR-recognizable if and only if it is closed (cf. [1] and [9]).

Now we need to specify some details regarding particular operations on tree languages. The *$\sigma$-product* of $\Sigma X_n$-tree languages $T_1, \ldots, T_m$ is the tree language $\sigma(T_1, \ldots, T_m) = \{\sigma(t_1, \ldots, t_m) \mid t_i \in T_i,\ 1 \leq i \leq m\}$, where $m \geq 1$ and $\sigma \in \Sigma_m$. We assume that the reader is already familiar with the operations of union, $x$-product and $x$-iteration. In the rest of this paper, we will use the operation of $x$-product in right-to-left manner, that is, for any tree languages $S, T \subseteq T_\Sigma(X_n)$ the $x$-product $T \cdot_x S$ is interpreted as a tree language in which the trees are obtained by taking a tree $s$ from $S$ and replacing every leaf symbol $x$ in $s$ by a tree from $T$. Different occurrences of $x$ may be replaced by different trees from $T$. We will also assume that $T \cdot_y R \cdot_x S$ always means $T \cdot_y (R \cdot_x S)$ for any tree languages $S, R, T \subseteq T_\Sigma(X_n)$ and variables $x, y \in X_n$.

Let $\Sigma$ be a ranked alphabet, and let $X_n$ be a set of variables. The set $RE(\Sigma X_n)$ of all *regular $\Sigma X_n$-expressions* and the tree language $T(\eta)$ represented by $\eta \in RE(\Sigma X_n)$ are defined in parallel as follows:

- $\emptyset \in RE(\Sigma X_n)$, $\quad T(\emptyset) = \emptyset$,
- $\forall x \in X_n : x \in RE(\Sigma X_n)$, $\quad T(x) = \{x\}$,

If $\sigma \in \Sigma_m$, $\eta_1, \eta_2, \ldots, \eta_m \in RE(\Sigma X_n)$, $x \in X_n$, then

- $(\eta_1) + (\eta_2) \in RE(\Sigma X_n)$, $\quad T((\eta_1) + (\eta_2)) = T(\eta_1) \cup T(\eta_2)$,
- $(\eta_2) \cdot_x (\eta_1) \in RE(\Sigma X_n)$, $\quad T((\eta_2) \cdot_x (\eta_1)) = T(\eta_2) \cdot_x T(\eta_1)$,
- $(\eta_1)^{*,x} \in RE(\Sigma X_n)$, $\quad T((\eta_1)^{*,x}) = T(\eta_1)^{*,x}$,
- $\sigma(\eta_1, \ldots, \eta_m) \in RE(\Sigma X_n)$, $\quad T(\sigma(\eta_1, \ldots, \eta_m)) = \sigma(T(\eta_1), \ldots, T(\eta_m))$.

Some parentheses can be omitted from regular $\Sigma X_n$-expressions, if a precedence relation is assumed between the operations of $\sigma$-product, $x$-iteration, $x$-product, and union in the given order.

A regular $\Sigma X_n$-expression $\zeta$ is *subexpression* of $\eta$ if $\zeta$ occurs in the inductive definition of $\eta$. The set of all subexpressions of $\eta$ will be denoted by *Sub($\eta$)*. The operation omission on regular $\Sigma X_n$-expressions is defined as follows: Let us consider $\sigma \in \Sigma_m$, $x \in X$, $\eta_1, \eta_2, \ldots, \eta_m \in RE(\Sigma X_n)$ and the regular $\Sigma X_n$-expressions $(\eta_1) + (\eta_2)$, $(\eta_2) \cdot_x (\eta_1)$, $(\eta_1)^{*,x}$ and $\sigma(\eta_1, \ldots, \eta_m)$. By omitting $\eta_1$ from them we get $\eta_2$, $\eta_2$, $\eta_1$ and $\sigma(\zeta, \eta_2, \ldots, \eta_m)$ respectively, where $\zeta$ is a variable occurring in $T(\eta_1)$, if such exists, otherwise $\zeta = \emptyset$. We allow the omission of $\eta_1$ from $(\eta_1)^{*,x}$ to result in $x$ as well. If we omit $\eta_2$ from $(\eta_1) + (\eta_2)$ and $(\eta_2) \cdot_x (\eta_1)$ we get $\eta_1$

and $\eta_1$ respectively. Omission on regular $\Sigma X_n$-expressions is not well-defined, but we do not need it to be so. Let $\eta$ be a regular $\Sigma X_n$-expression, and let $\zeta$ be a subexpression of $\eta$. We call $\zeta$ *redundant in* $\eta$, if $\zeta$ can be omitted from $\eta$ so that $T(\eta)$ remains unchanged after omission. A regular $\Sigma X_n$-expression is *reduced* if it has no redundant subexpressions. As in the string case, a regular $\Sigma X_n$-expression may have several different reduced forms.

Now we adapt the concept of *iterational height* for tree languages which will be used to identify the length of the longest $x$-path that will be used in an $x$-iteration of a particular tree language. Let $x \in X$ be a variable, and let $\eta$ be a regular $\Sigma X_n$-expression in form $(\zeta)^{*,x}$. The *iterational height* of $x$ in $\eta$ ($ih_x(\eta)$ for short) is defined as $max\{|u| : u \in g_x(T(\zeta))\}$, if $g_x(T(\zeta))$ is finite. If $g_x(T(\zeta))$ is infinite, then let $ih_x(\eta)$ be the infinity $\infty$ that we will treat as the largest integer. Let now $\eta$ be a reduced regular $\Sigma X_n$-expression in any form. We define $ih_x(\eta)$ as $max\{ih_x((\zeta)^{*,x}) \mid (\zeta)^{*,x} \in Sub(\eta)\}$, if $Sub(\eta)$ contains an expression in form $(\zeta)^{*,x}$, and 0 otherwise. The iterational height of $x$ in a regular tree language $T$ ($ih_x(T)$ for short) is defined as $min\{ih_x(\eta) \ : \ T = T(\eta)\}$.

**Example 7.** Let $\Sigma = \Sigma_2 = \{\sigma\}$ and $X = \{x, y\}$ hold and let us consider the regular $\Sigma X$-expression $\zeta = \sigma(y, \sigma(y, x)) + \sigma(y, \sigma(y, \sigma(y, x)))$. It is easy to see that $ih_x((\zeta)^{*,x}) = 3$. Taking $\eta = \sigma(y, x) + (\zeta)^{*,x}$ we have $ih_x(\eta) = 3$ because $\eta$ has a subexpression in form $(\zeta)^{*,x}$ for which $ih_x((\zeta)^{*,x}) = 3$. Considering the tree language $T(\eta)$ we get $ih_x(T(\eta)) = 1$, because $T(\eta)$ can be represented also by $(\sigma(y, x))^{*,x}$, for which $ih_x((\sigma(y, x))^{*,x}) = 1$.

**Lemma 8.** *Let $\eta$ be a reduced regular $\Sigma X_n$-expression of the form $(\zeta)^{*,x}$. If $T(\eta)$ is a monotone DR-language, then $ih_x(T(\eta)) \leq 1$.*

*Proof.* Let $\eta$ be a reduced regular $\Sigma X_n$-expression of the form $(\zeta)^{*,x}$, and let $\mathfrak{A}$ be a monotone DR-recognizer which recognizes $T(\eta)$ with the partial ordering $\leq$. Without the loss of generality we can suppose that $\mathfrak{A}$ is reduced and normalized, thus there is exactly one state $a \in A$ for which $a \in \alpha(x)$ and $au = a$ hold for every word $u \in g_x(T(\zeta))$. Since $\mathfrak{A}$ is monotone, we see that $a\sigma = a$ for any letter $\sigma$ that is present in any of the words of $g_x(T(\zeta))$. Moreover, there is no state $a' \in A \setminus \{a\}$ for which $a \leq a'$ and $a' \in \alpha(x)$, and there is no state $a'' \in \alpha(x) \setminus \{a\}$ for which $a'' \leq a$ and $a''\sigma = a''$ hold for every letter $\sigma$ that is present in any of the words of $g_x(T(\zeta))$. Hence $\eta$ can be written in form $(\zeta'')^{*,x} \cdot_x \zeta'$, where $\zeta'$ represents the tree language that $\mathfrak{A}$ recognizes by taking $A^{(i)} = \{a\}$, and leaving $A^{(j)}$ unchanged if $j \neq i$, and where $\zeta''$ is the representation of the trees that we can get by decomposition of every tree $t \in T(\zeta)$ at every point of the paths in $g_x(t)$. It is easy to see that $T(\eta) = T((\zeta'')^{*,x} \cdot_x \zeta')$ and $ih_x(T((\zeta'')^{*,x} \cdot_x \zeta')) = 1$, that is $ih_x(T(\eta)) \leq 1$. □

# 5 A simple characterization

Let $\mathfrak{A} = (\mathcal{A}, a_0, \mathbf{a})$ be a monotone DR $\Sigma X_n$-recognizer, where $\mathcal{A} = (A, \Sigma^A)$, $A = \{a_0, \ldots, a_k\}$ and $\mathbf{a} = (A^{(1)}, \ldots, A^{(n)})$. Without the loss of generality we

can suppose that $a_0 \leq a_1 \leq \ldots \leq a_k$ holds. Let $\Xi_k = \{\xi_0, \ldots, \xi_k\}$ be a set of auxiliary variables for which $X_n \cap \Xi_k = \emptyset$ holds. Furthermore, let $\phi : A \to \Xi_k$ be a bijective mapping defined by $\phi(a_i) = \xi_i$ $(0 \leq i \leq k)$. Now we construct the regular $\Sigma(X_n \cup \Xi_k)$-expression $\eta$ as follows:

$$\eta = \eta_k \cdot_{\xi_k} \eta_{k-1} \cdot_{\xi_{k-1}} \cdots \cdot_{\xi_1} \eta_0,$$

where for each $i = 0, \ldots, k$

$$\eta_i = (p_1^i + \cdots + p_{l_i}^i + y_1^i + \cdots + y_{r_i}^i) \cdot_{\xi_i} (t_1^i + \cdots + t_{j_i}^i)^{*,\xi_i},$$

and where

1) $y_1^i, \ldots, y_{r_i}^i$ are all the elements of the set $\{x_z \in X_n |\ a_i \in A^{(z)}\}$,

2) $p_s^i = \sigma(\xi_{i_1}, \ldots, \xi_{i_m})$ for such $\sigma \in \Sigma_m$ and $\xi_{i_v} \in \Xi_k$ $(1 \leq v \leq m)$ that $\sigma(a_i) = (\phi^{-1}(\xi_{i_1}), \ldots, \phi^{-1}(\xi_{i_m}))$ and $a_i \notin \bigcup_{1 \leq v \leq m} \{\pi_v(\sigma(a_i))\}$ hold $(1 \leq s \leq l_i)$,

3) $t_s^i = \sigma(\xi_{i_1}, \ldots, \xi_{i_m})$ for such $\sigma \in \Sigma_m$ and $\xi_{i_v} \in \Xi_k$ $(1 \leq v \leq m)$ that $\sigma(a_i) = (\phi^{-1}(\xi_{i_1}), \ldots, \phi^{-1}(\xi_{i_m}))$ and $a_i \in \bigcup_{1 \leq v \leq m} \{\pi_v(\sigma(a_i))\}$ hold $(1 \leq s \leq j_i)$,

4) $|\ \{p_1^i, \ldots, p_{l_i}^i\}\ | + |\ \{t_1^i, \ldots, t_{j_i}^i\}\ | = |\ \Sigma\ |$.

The regular $\Sigma(X_n \cup \Xi_k)$-expression $\eta$ constructed above is called the *trivial regular expression belonging to* $\mathfrak{A}$, and is denoted by $\eta_{\mathfrak{A}}$. We use the word *trivial* because $\eta_{\mathfrak{A}}$ describes $T(\mathfrak{A})$ by its computation in $\mathfrak{A}$, where for every $0 \leq i \leq k$, $\eta_i$ is responsible for the computation starting in state $a_i$. That part of $\eta_i$ which is iterated by the operation $*,\xi_i$ is called the *iterating part* of $\eta_i$, and the part of $\eta_i$ which is inserted by $\cdot_{\xi_i}$ product into the variables $\xi_i$ of the iterating part is called the *terminating part* of $\eta_i$. We will call the expressions of the form $\eta_k \cdot_{\xi_k} \ldots \eta_1 \cdot_{\xi_1} \eta_0$ by *chains*.

Let the $a_0 \leq a_1 \leq \ldots \leq a_k$ linear ordering hold on the state set of the monotone DR $\Sigma X_n$-recognizer $\mathfrak{A}$. Let us define the DR $\Sigma X_n$-recognizer $\mathfrak{A}_i$ as follows: $\mathfrak{A}_i = (\mathcal{A}_i, a_i, \mathbf{a}_i)$, where $\mathcal{A}_i = (A \cap \{a_i, \ldots, a_k\}, \Sigma^A)$, and $\mathbf{a}_i = (A^{(1)} \cap \{a_i, \ldots, a_k\}, \ldots, A^{(n)} \cap \{a_i, \ldots, a_k\})$. It is obvious that $\mathfrak{A}_i$ recognizes $T(\mathfrak{A}, a_i)$.

**Lemma 9.** *For a monotone DR $\Sigma X_n$-recognizer $\mathfrak{A}$ the equality $T(\mathfrak{A}) = T(\eta_{\mathfrak{A}})$ holds.*

*Proof.* Let $\mathfrak{A}$ be a monotone DR $\Sigma X_n$-recognizer, and let $\eta_{\mathfrak{A}}$ be the trivial regular expression belonging to $\mathfrak{A}$. Let us also suppose that $\mathfrak{A} = (\mathcal{A}, a_0, \mathbf{a})$, $\mathcal{A} = (A, \Sigma)$, $A = \{a_0, \ldots, a_k\}$, and the linear ordering $a_0 \leq \ldots \leq a_k$ holds on $A$. The proof is continued by induction on the number of states in $\mathfrak{A}$.

If $k = 0$, then $T(\mathfrak{A}) = T_\Sigma(X_n \cap \{x_i |\ a_0 \in A^{(i)}\})$ holds because $A$ is singleton. Obviously $\eta_{\mathfrak{A}} = \eta_0$ holds, too. By the definition of $\eta_{\mathfrak{A}}$, every $\sigma \in \Sigma$ is present in the iterating part of $\eta_0$, and every $x \in \{x_i |\ a_0 \in A^{(i)}\} \subseteq X_n$ is present in the terminating part of $\eta_0$. Hence, $T(\eta_{\mathfrak{A}}) = T_\Sigma(X_n \cap \{x_i |\ a_0 \in A^{(i)}\})$, that is, $T(\mathfrak{A}) = T(\eta_{\mathfrak{A}})$.

Let us now suppose as our induction hypothesis that $T(\mathfrak{A}_i) = \eta_k \cdot_{\xi_k} \ldots \cdot_{\xi_{i+1}} \eta_i$ holds for every $1 \leq i \leq k$. Now we construct the $\Sigma(X_n \cup \Xi_k)$-recognizer $\mathfrak{A}'$ as follows: $\mathfrak{A}' = (\mathcal{A}, a_0, \mathbf{a}')$, where $\mathbf{a}' = (A^{(1)} \cap \{a_0\}, \ldots, A^{(n)} \cap \{a_0\}, \{a_0\}, \ldots, \{a_k\}) \in \mathfrak{p}(A)^{n+k+1}$. To interpret the meaning of $T(\mathfrak{A}')$ let us treat $X_n \cup \Xi_k$ as the set $X_{n+k+1}$, where $x_{n+i+1} = \xi_i$, and let the mapping $\alpha$ be defined as $\alpha(\xi_i) = \alpha(x_{n+i+1}) = A^{(n+i+1)}$ $(i = 0, \ldots, k)$.

It can be easily seen that $T(\mathfrak{A}) = T(\mathfrak{A}_k) \cdot_{\xi_k} \ldots \cdot_{\xi_2} T(\mathfrak{A}_1) \cdot_{\xi_1} T(\mathfrak{A}')$, and $T(\mathfrak{A}') = T(\eta_0)$. Hence

$$\begin{aligned} T(\mathfrak{A}) &= T(\mathfrak{A}_k) \cdot_{\xi_k} T(\mathfrak{A}_{k-1}) \cdot_{\xi_{k-1}} \ldots \cdot_{\xi_2} T(\mathfrak{A}_1) \cdot_{\xi_1} T(\mathfrak{A}') && = \\ &= T(\eta_k) \cdot_{\xi_k} T(\eta_k \cdot_{\xi_k} \eta_{k-1}) \cdot_{\xi_{k-1}} \ldots \cdot_{\xi_2} T(\eta_k \cdot_{\xi_k} \ldots \cdot_{\xi_2} \eta_1) \cdot_{\xi_1} T(\eta_0) && = \\ &= T(\eta_k) \cdot_{\xi_k} T(\eta_{k-1}) \cdot_{\xi_{k-1}} \ldots \cdot_{\xi_2} T(\eta_1) \cdot_{\xi_1} T(\eta_0) && = \\ &= T(\eta_k \cdot_{\xi_k} \eta_{k-1} \cdot_{\xi_{k-1}} \ldots \cdot_{\xi_2} \eta_1 \cdot_{\xi_1} \eta_0) && = \\ &= T(\eta). \end{aligned}$$

□

# 6 Remarks on the decomposition of $\eta$

In this section we give some remarks on the decomposition of the regular $\Sigma(X_n \cup \Xi_k)$-expression $\eta = \eta_k \cdot_{\xi_k} \ldots \eta_1 \cdot_{\xi_1} \eta_0$. If there is at most one symbol in the terminating part of $\eta_i$, then the decomposition in the $\eta_i$ part makes no sense, hence we assume in this section that there are at least two symbols in the terminating part of $\eta_i$.

We say that $\eta = \eta_k \cdot_{\xi_k} \ldots \cdot_{\xi_{i+1}} \eta_i \cdot_{\xi_i} \ldots \cdot_{\xi_1} \eta_0$ can be decomposed in the $\eta_i$ part if it can be given in the form

$$\eta = \eta_k \cdot_{\xi_k} \ldots \cdot_{\xi_{i+1}} \eta_i \cdot_{\xi_i} \ldots \cdot_{\xi_1} \eta_0 =$$

$$\eta_k \cdot_{\xi_k} \ldots \cdot_{\xi_{i+1}} (p_1^i + \ldots + p_{l_i}^i + y_1^i + \ldots + y_{r_i}^i) \cdot_{\xi_i} (t_1^i + \ldots + t_{j_i}^i)^{*,\xi_i} \cdot_{\xi_i} \ldots \cdot_{\xi_1} \eta_0 =$$

$$\eta_k \cdot_{\xi_k} \ldots \cdot_{\xi_{i+1}} (y_1^i) \cdot_{\xi_i} (t_1^i + \ldots + t_{j_i}^i)^{*,\xi_i} \cdot_{\xi_i} \ldots \cdot_{\xi_1} \eta_0 +$$

$$\vdots$$

$$+ \eta_k \cdot_{\xi_k} \ldots \cdot_{\xi_{i+1}} (y_{r_i}^i) \cdot_{\xi_i} (t_1^i + \ldots + t_{j_i}^i)^{*,\xi_i} \cdot_{\xi_i} \ldots \cdot_{\xi_1} \eta_0 +$$

$$+ \eta_k \cdot_{\xi_k} \ldots \cdot_{\xi_{i+1}} (p_1^i) \cdot_{\xi_i} (t_1^i + \ldots + t_{j_i}^i)^{*,\xi_i} \cdot_{\xi_i} \ldots \cdot_{\xi_1} \eta_0 +$$

$$\vdots$$

$$+ \eta_k \cdot_{\xi_k} \ldots \cdot_{\xi_{i+1}} (p_{l_i}^i) \cdot_{\xi_i} (t_1^i + \ldots + t_{j_i}^i)^{*,\xi_i} \cdot_{\xi_i} \ldots \cdot_{\xi_1} \eta_0,$$

where

(i) $y_s^i \in X_n$ $(1 \leq s \leq r_i,\ 0 \leq r_i \leq n)$,

(ii) $p_s^i = \sigma(\xi_{i_1}, \ldots, \xi_{i_m})$, for some $\sigma \in \Sigma_m$, $\xi_{i_v} \in \Xi_k$, $1 \leq v \leq m$, $1 \leq s \leq l_i$,

(iii) $t_s^i = \sigma(\xi_{i_1}, \ldots, \xi_{i_m})$, for some $\sigma \in \Sigma_m$, $\xi_{i_v} \in \Xi_k$, $1 \leq v \leq m$, $1 \leq s \leq j_i$.

Now we state a necessary condition for the existence of such decompositions.

**Lemma 10.** *The expression $\eta = \eta_k \cdot_{\xi_k} \dots \eta_1 \cdot_{\xi_1} \eta_0$ can be decomposed in the $\eta_i$ part, if every operational symbol in the iterating part of $\eta_i$ contains the auxiliary variable $\xi_i$ at most once among its leaves.*

*Proof.* Let us suppose that the condition of the lemma holds. Let us denote in this proof the regular $\Sigma(X_n \cup \Xi_k)$-expressions $\eta_k \cdot_{\xi_k} \dots \cdot_{\xi_{i+2}} \eta_{i+1}$ and $(t_1^i + \dots + t_{j_i}^i)^{*,\xi_i} \cdot_{\xi_i} \dots \cdot_{\xi_1} \eta_0$ by $\zeta''$ and $\zeta'$, respectively. It is easy to see that for every tree $t \in T(\zeta')$ the set $g_{\xi_i}(t)$ is a singleton or the empty set. By the definition of the $x$-product of tree languages, using the condition of the lemma, we get

$$T(\eta) = T(\zeta'' \cdot_{\xi_{i+1}} (p_1^i + \dots + p_{l_i}^i + y_1^i + \dots + y_{r_i}^i) \cdot_{\xi_i} \zeta') =$$

$$= T(\zeta'') \cdot_{\xi_{i+1}} T(p_1^i + \dots + p_{l_i}^i + y_1^i + \dots + y_{r_i}^i) \cdot_{\xi_i} T(\zeta') =$$

$$= T(\zeta'') \cdot_{\xi_{i+1}} ( T(p_1^i) \cdot_{\xi_i} T(\zeta') \cup \dots \cup T(p_{l_i}^i) \cdot_{\xi_i} T(\zeta') \cup T(y_1^i) \cdot_{\xi_i} T(\zeta') \cup \dots$$

$$\dots \cup T(y_{r_i}^i) \cdot_{\xi_i} T(\zeta') ) =$$

$$= T(\zeta'' \cdot_{\xi_{i+1}} p_1^i \cdot_{\xi_i} \zeta' + \dots + \zeta'' \cdot_{\xi_{i+1}} p_{l_i}^i \cdot_{\xi_i} \zeta' + \zeta'' \cdot_{\xi_{i+1}} y_1^i \cdot_{\xi_i} \zeta' + \dots + \zeta'' \cdot_{\xi_{i+1}} y_{r_i}^i \cdot_{\xi_i} \zeta').$$

Hence the decomposition in $\eta_i$ led to an equivalent regular $\Sigma(X_n \cup \Xi_k)$-expression. □

It is clear that if the auxiliary variable $\xi_i$ does not occur in the subexpression $\eta_{i-1} \cdot_{\xi_{i-1}} \dots \eta_1 \cdot_{\xi_1} \eta_0$, then the factor $\eta_i$ can be omitted from the expression of $\eta$. Let us note that the decomposed parts will also be called *chains*, that is, the above mentioned chain $\eta$ is decomposed into finite union of chains.

The variables $y_1^i, \dots, y_{r_i}^i$ can be left in any of the decomposed chains, because by inserting these variables into the iterating part during the $\xi_i$-product we terminate that path, that is, no auxiliary variable can be reached after from these variables.

Now we state the converse of the Lemma 10.

**Lemma 11.** *If the expression $\eta = \eta_k \cdot_{\xi_k} \dots \eta_1 \cdot_{\xi_1} \eta_0$ can be decomposed in the $\eta_i$ part, then every operational symbol in the iterating part of $\eta_i$ contains the auxiliary variable $\xi_i$ at most once among its leaves.*

*Proof.* Let us suppose that there is an operational symbol $\sigma \in \Sigma_m$ in the iterating part of the decomposed $\eta_i$, where $\xi_i$ occurs at least twice among the leaves of $\sigma$. Let $\zeta''$ and $\zeta'$ stand for the regular $\Sigma(X_n \cup \Xi_k)$-expressions $\eta_k \cdot_{\xi_k} \dots \cdot_{\xi_{i+2}} \eta_{i+1}$ and $\eta_{i-1} \cdot_{\xi_{i-1}} \dots \cdot_{\xi_1} \eta_0$, respectively. For the sake of simplicity we will write $\tilde{\sigma}(\xi_i, \xi_i)$ instead of $\sigma(\xi_1', \dots, \xi_{v_1}', \xi_i, \xi_1'', \dots, \xi_{v_2}'', \xi_i, \xi_1''', \dots, \xi_{v_3}''')$, where $v_1, v_2, v_3 \in \{0, 1, \dots, m-2\}$, $v_1 + v_2 + v_3 = m - 2$, and $\xi_{z'}', \xi_{z''}'', \xi_{z'''}''' \in \Xi_k$, $(z' \in \{1, \dots, v_1\}$, $z'' \in \{1, \dots, v_2\}$, $z''' \in \{1, \dots, v_3\})$. It is obvious that $T(\zeta'' \cdot_{\xi_{i+1}} (p_1^i + \dots + p_{l_i}^i + y_1^i + \dots + y_{r_i}^i) \cdot_{\xi_i} \tilde{\sigma}(\xi_i, \xi_i) \cdot_{\xi_i} \zeta') \subset T(\eta)$. Moreover, $T(\zeta'' \cdot_{\xi_{i+1}} \tilde{\sigma}(s_1, s_2) \cdot_{\xi_i} \zeta') \subset T(\eta)$ holds too for every different pair of symbols $s_1, s_2 \in \{p_1^i, \dots, p_{l_i}^i, y_1^i, \dots, y_{r_i}^i\}$. On the other hand $T(\zeta'' \cdot_{\xi_{i+1}} \tilde{\sigma}(s_1, s_2) \cdot_{\xi_i} \zeta') \not\subseteq \bigcup_{1 \le v \le l_i} T(\zeta'' \cdot_{\xi_{i+1}} \tilde{\sigma}(p_v^i, p_v^i) \cdot_{\xi_i} \zeta') \cup \bigcup_{1 \le v \le r_i} T(\zeta'' \cdot_{\xi_{i+1}} \tilde{\sigma}(y_v^i, y_v^i) \cdot_{\xi_i} \zeta')$, which is a contradiction because there are such trees in $T(\eta)$ which are not present in the decomposed chains of $\eta$. □

The above results can be summarized in

**Theorem 12.** *The expression* $\eta = \eta_k \cdot_{\xi_k} \ldots \eta_1 \cdot_{\xi_1} \eta_0$ *can be decomposed in the* $\eta_i$ *part if and only if every operational symbol in the iterating part of* $\eta_i$ *contains the auxiliary variable* $\xi_i$ *at most once among its leaves.*

# 7 Remarks on the number of the auxiliary variables in $\eta_{\mathfrak{A}}$

In this section we deal with the number of the auxiliary variables in $\eta_{\mathfrak{A}}$. We will also give some methods by which this number can be possibly reduced. It is obvious that if the number of states is $k$, then the representation can be done with $k+1$ auxiliary variables.

It is said that we *terminate a variable* $x \in X_n$ *in a tree* $t \in T_\Sigma(X_n)$ by a tree $p \in T_\Sigma(X_n)$, if the variable $x$ is not present among the leaves of the trees $p \cdot_x t$. Let $\zeta$ be a regular $\Sigma X_n$-expression. It is said that *a variable* $x \in X_n$ *is terminated in* $\zeta$, if there is no variable $x$ among the leaves of the trees of $T(\zeta)$.

Obviously, the number of the necessary auxiliary variables can be possibly decreased if we decompose $\eta$ at every possible place (as seen in the previous section), and we renumber the auxiliary variables from 0 in each decomposed chain of $\eta$ separately.

It is clear that a variable $\xi_i$ is terminated in the $\eta_i$ part, that is the variable $\xi_i$ will not occur at any leaf from this point during the right-to-left evaluation of $\eta$. Hence we can reuse some auxiliary variables within a chain. Let us suppose that there is an auxiliary variable $\xi_j$ in the chain which has its first occurrence in the terminating part of $\eta_i$ (during the right-to-left evaluation of the chain). In this case every occurrence of $\xi_j$ in $\eta$ can be replaced with $\xi_i$, by which we have done an equivalent transformation. In fact, we can also use the elements of $X_n$ to decrease the number of the auxiliary variables. The idea is the same, that is, an existing auxiliary variable $\xi_i$ can be replaced with a variable $x$ if $\xi_i$ gets terminated before the first occurrence of $x$.

On the basis of the remarks above the following steps can possibly reduce the number of the auxiliary variables:

(i) decompose $\eta_{\mathfrak{A}}$ into union of as many chains as possible

(ii) decrease the number of the auxiliary variables in these decomposed chains separately

(iii) renumber the auxiliary variables starting with 0 in each chain

**Example 13.** Let $\mathfrak{A} = (\mathcal{A}, a_0, \mathbf{a})$ be a DR $\Sigma X_3$-recognizer, where $\mathcal{A} = (A, \Sigma)$, $A = \{a_0, a_1, a_2, a_3\}$, $\Sigma = \{\sigma_1, \sigma_2, \sigma_3\}$, $\sigma_i \in \Sigma_i$ $(1 \leq i \leq 3)$, and $\mathbf{a} = (\{a_0\}, \{a_0, a_2\}, \{a_1, a_2, a_3\})$. $\Sigma$ is realized in $\mathcal{A}$ as follows:

$$\begin{array}{lll}\sigma_1(a_0)=(a_1), & \sigma_2(a_0)=(a_0,a_1), & \sigma_3(a_0)=(a_0,a_0,a_1),\\ \sigma_1(a_1)=(a_3), & \sigma_2(a_1)=(a_2,a_2), & \sigma_3(a_1)=(a_1,a_3,a_3),\\ \sigma_1(a_2)=(a_3), & \sigma_2(a_2)=(a_2,a_3), & \sigma_3(a_2)=(a_2,a_3,a_3),\\ \sigma_1(a_3)=(a_3), & \sigma_2(a_3)=(a_3,a_3), & \sigma_3(a_3)=(a_3,a_3,a_3).\end{array}$$

The resulting regular expression is the following:

$$\eta_{\mathfrak{A}} = \eta_3 \cdot_{\xi_3} \eta_2 \cdot_{\xi_2} \eta_1 \cdot_{\xi_1} \eta_0 =$$

$$= (x_3) \cdot_{\xi_3} (\sigma_1(\xi_3) + \sigma_2(\xi_3,\xi_3) + \sigma_3(\xi_3,\xi_3,\xi_3))^{*,\xi_3} \cdot_{\xi_3}$$
$$\cdot_{\xi_3} (\sigma_1(\xi_3) + x_2 + x_3) \cdot_{\xi_2} (\sigma_2(\xi_2,\xi_3) + \sigma_3(\xi_2,\xi_3,\xi_3))^{*,\xi_2} \cdot_{\xi_2}$$
$$\cdot_{\xi_2} (\sigma_1(\xi_3) + \sigma_2(\xi_2,\xi_2) + x_3) \cdot_{\xi_1} (\sigma_3(\xi_1,\xi_3,\xi_3))^{*,\xi_1} \cdot_{\xi_1}$$
$$\cdot_{\xi_1} (\sigma_1(\xi_1) + x_1 + x_2) \cdot_{\xi_0} (\sigma_2(\xi_0,\xi_1) + \sigma_3(\xi_0,\xi_0,\xi_1))^{*,\xi_0}$$

We can decompose the above chain in the $\eta_1$ factor by which we get

$$(x_3) \cdot_{\xi_3} (\sigma_1(\xi_3) + \sigma_2(\xi_3,\xi_3) + \sigma_3(\xi_3,\xi_3,\xi_3))^{*,\xi_3} \cdot_{\xi_3}$$
$$\cdot_{\xi_3} (\sigma_1(\xi_3) + x_2 + x_3) \cdot_{\xi_2} (\sigma_2(\xi_2,\xi_3) + \sigma_3(\xi_2,\xi_3,\xi_3))^{*,\xi_2} \cdot_{\xi_2}$$
$$\cdot_{\xi_2} (\sigma_1(\xi_3) + x_3) \cdot_{\xi_1} (\sigma_3(\xi_1,\xi_3,\xi_3))^{*,\xi_1} \cdot_{\xi_1}$$
$$\cdot_{\xi_1} (\sigma_1(\xi_1) + x_1 + x_2) \cdot_{\xi_0} (\sigma_2(\xi_0,\xi_1) + \sigma_3(\xi_0,\xi_0,\xi_1))^{*,\xi_0}$$
$$+$$
$$(x_3) \cdot_{\xi_3} (\sigma_1(\xi_3) + \sigma_2(\xi_3,\xi_3) + \sigma_3(\xi_3,\xi_3,\xi_3))^{*,\xi_3} \cdot_{\xi_3}$$
$$\cdot_{\xi_3} (\sigma_1(\xi_3) + x_2 + x_3) \cdot_{\xi_2} (\sigma_2(\xi_2,\xi_3) + \sigma_3(\xi_2,\xi_3,\xi_3))^{*,\xi_2} \cdot_{\xi_2}$$
$$\cdot_{\xi_2} (\sigma_2(\xi_2,\xi_2)) \cdot_{\xi_1} (\sigma_3(\xi_1,\xi_3,\xi_3))^{*,\xi_1} \cdot_{\xi_1}$$
$$\cdot_{\xi_1} (\sigma_1(\xi_1) + x_1 + x_2) \cdot_{\xi_0} (\sigma_2(\xi_0,\xi_1) + \sigma_3(\xi_0,\xi_0,\xi_1))^{*,\xi_0}$$

Simplifying the above expression we can write

$$(x_3) \cdot_{\xi_3} (\sigma_1(\xi_3) + \sigma_2(\xi_3,\xi_3) + \sigma_3(\xi_3,\xi_3,\xi_3))^{*,\xi_3} \cdot_{\xi_3}$$
$$\cdot_{\xi_3} (\sigma_1(\xi_3) + x_3) \cdot_{\xi_1} (\sigma_3(\xi_1,\xi_3,\xi_3))^{*,\xi_1} \cdot_{\xi_1}$$
$$\cdot_{\xi_1} (\sigma_1(\xi_1) + x_1 + x_2) \cdot_{\xi_0} (\sigma_2(\xi_0,\xi_1) + \sigma_3(\xi_0,\xi_0,\xi_1))^{*,\xi_0}$$
$$+$$
$$(x_3) \cdot_{\xi_3} (\sigma_1(\xi_3) + \sigma_2(\xi_3,\xi_3) + \sigma_3(\xi_3,\xi_3,\xi_3))^{*,\xi_3} \cdot_{\xi_3}$$
$$\cdot_{\xi_3} (\sigma_1(\xi_3) + x_2 + x_3) \cdot_{\xi_2} (\sigma_2(\xi_2,\xi_3) + \sigma_3(\xi_2,\xi_3,\xi_3))^{*,\xi_2} \cdot_{\xi_2}$$
$$\cdot_{\xi_2} (\sigma_2(\xi_2,\xi_2)) \cdot_{\xi_1} (\sigma_3(\xi_1,\xi_3,\xi_3))^{*,\xi_1} \cdot_{\xi_1}$$
$$\cdot_{\xi_1} (\sigma_1(\xi_1) + x_1 + x_2) \cdot_{\xi_0} (\sigma_2(\xi_0,\xi_1) + \sigma_3(\xi_0,\xi_0,\xi_1))^{*,\xi_0}$$

Reusing the variables $(\xi_0 \to \xi_3)$ and $(x_3 \to \xi_1)$ in the above chains we get

$$(x_3) \cdot_{\xi_0} (\sigma_1(\xi_0) + \sigma_2(\xi_0, \xi_0) + \sigma_3(\xi_0, \xi_0, \xi_0))^{*,\xi_0} \cdot_{\xi_0}$$

$$\cdot_{\xi_0} (\sigma_1(\xi_0) + x_3) \cdot_{x_3} (\sigma_3(x_3, \xi_0, \xi_0))^{*,x_3} \cdot_{x_3}$$

$$\cdot_{x_3} (\sigma_1(x_3) + x_1 + x_2) \cdot_{\xi_0} (\sigma_2(\xi_0, x_3) + \sigma_3(\xi_0, \xi_0, x_3))^{*,\xi_0}$$

$$+$$

$$(x_3) \cdot_{\xi_0} (\sigma_1(\xi_0) + \sigma_2(\xi_0, \xi_0) + \sigma_3(\xi_0, \xi_0, \xi_0))^{*,\xi_0} \cdot_{\xi_0}$$

$$\cdot_{\xi_0} (\sigma_1(\xi_0) + x_2 + x_3) \cdot_{\xi_2} (\sigma_2(\xi_2, \xi_0) + \sigma_3(\xi_2, \xi_0, \xi_0))^{*,\xi_2} \cdot_{\xi_2}$$

$$\cdot_{\xi_2} (\sigma_2(\xi_2, \xi_2)) \cdot_{x_3} (\sigma_3(x_3, \xi_0, \xi_0))^{*,x_3} \cdot_{x_3}$$

$$\cdot_{x_3} (\sigma_1(x_3) + x_1 + x_2) \cdot_{\xi_0} (\sigma_2(\xi_0, x_3) + \sigma_3(\xi_0, \xi_0, x_3))^{*,\xi_0}$$

We can see that the initial number of the auxiliary variables is reduced from 4 to 2.

We finish the discussion of the section with

**Lemma 14.** *If $\Sigma = \Sigma_1$, then for any monotone DR $\Sigma X_n$-recognizer $\mathfrak{A}$ one auxiliary variable is enough to represent $\eta_{\mathfrak{A}}$.*

*Proof.* Let $\Sigma = \Sigma_1$, and let $\eta_{\mathfrak{A}}$ be the $\Sigma(X_n \cup \Xi_k)$-regular expression belonging to $\mathfrak{A}$. As we have only unary operational symbols, $\xi_i$ occurs at most once among the leaves of an operational symbol from the iterating part of each $\eta_i$. So $\eta$ can be decomposed into finite union of chains, moreover, the decomposition can be done at each $\eta_i$ factor. The condition $\Sigma = \Sigma_1$ implies also that during the evaluation at every step there is exactly one auxiliary variable which is not terminated. Since the variable $\xi_0$ gets terminated in the terminating part of $\eta_0$, we can reuse $\xi_0$ instead of introducing a new auxiliary variable. Continuing the idea we can rewrite all decomposed chains so that they will use only $\xi_0$ as an auxiliary variable. □

# 8 Characterization of monotone DR-languages

It is a well-known fact that the class of DR-languages is closed under $\sigma$-products, but not under union, $x$-product, and $x$-iteration. It means that the $x$-product, $x$-iteration and union of monotone DR-languages are not always deterministic (cf. [3] and [8]). Conversely, using the three operations mentioned above on not closed languages can result in a closed (or even monotone) DR-languages, as it can be seen from the examples below.

**Example 15.** Let us consider the regular tree languages $S = \{\sigma(x, x),\ \sigma(y, y)\}$ and $T = \{\sigma(x, y),\ \sigma(y, x)\}$. It is clear that they are not closed, but the tree language $S \cup T = \{\sigma(x, x),\ \sigma(y, y),\ \sigma(x, y),\ \sigma(y, x)\}$ is closed, that is, DR-recognizable. Moreover, $S \cup T$ is monotone.

**Example 16.** Let us now consider the regular tree languages $S = \{z,\ \sigma(x,x),\ \sigma(y,y)\}$ and $T = \{\sigma(x,y),\ \sigma(y,x)\}$. They are not closed, but the tree language $T \cdot_z S = \{\sigma(x,x),\ \sigma(y,y),\ \sigma(x,y),\ \sigma(y,x)\}$ is DR-recognizable, and what is more, $T \cdot_z S$ is monotone.

**Example 17.** Let $S$ be the following regular tree language: $S = \{\sigma(x,\sigma(x,y)),\ \sigma(x,\sigma(y,x)),\ \sigma(x,x),\ \sigma(y,y),\ \sigma(x,y),\ \sigma(y,x)\}$. $S$ is not closed, but the tree language $(S)^{*,x}$ is closed, moreover, $(S)^{*,x}$ is monotone.

Let $S \subseteq T_\Sigma(X_n)$ be a tree language and let $p \in T_\Sigma(X_n)$ be a tree. The *root* $\mathrm{root}(p)$, *leaves* $\mathrm{leaves}(p)$ and the set of *subtrees* $\mathrm{Sub}(p)$ of the tree $p$ are defined as follows:

(i) If $p \in X_n$, then $root(p) = p$, $leaves(p) = \{p\}$ and $Sub(p) = \{p\}$.

(ii) If $p = \sigma(t_1,\ldots,t_m)$, $\sigma \in \Sigma_m$, $t_i \in T_\Sigma(X_n)$, $1 \le i \le m$, then $root(p) = \sigma$, $leaves(p) = \bigcup_{1\le i\le m} leaves(t_i)$, and $Sub(p) = \{p\} \cup \bigcup_{1\le i\le m}(Sub(t_i))$.

The above functions are extended from trees to tree languages as follows: $root(S) = \{root(p) \mid p \in S\}$, $leaves(S) = \bigcup_{p\in S} leaves(p)$, and $Sub(S) = \bigcup_{p\in S} Sub(p)$.

Let $\Sigma_S$ denote the set of *operational symbols appearing in* $S$, and is defined as $\Sigma_S = root(Sub(S)) \setminus X_n$. Let $\Sigma_{S,x}$ denote the set $\{\ \sigma \in \Sigma \mid \exists u \in g_x(S),\ \exists v \in \hat{\Sigma}^*,\ \exists z \in X_n\ :\ uv \in g_z(S),\ v = (\sigma,i)\ldots(\omega,j),\ \omega \in \Sigma,\ i,j \in N\}$.

Now we give a condition by which the $x$-product of two monotone DR-languages is also monotone.

**Theorem 18.** *Let $S,T \subseteq T_\Sigma(X_n)$ be monotone DR-languages, $x_i \in X_n$. If $\Sigma_{S,x_i} \cap root(T) = \emptyset$, then $T \cdot_{x_i} S$ is monotone.*

*Proof.* Assume that the conditions of the theorem hold. Let $\mathfrak{A} = (\mathcal{A}, a_0, \mathbf{a})$ and $\mathfrak{B} = (\mathcal{B}, b_0, \mathbf{b})$ be monotone DR $\Sigma X_n$-recognizers, where $\mathcal{A} = (A, \Sigma^{\mathcal{A}})$, $A = \{a_0,\ldots,a_k\}$, $\mathbf{a} = (A^{(1)},\ldots,A^{(n)})$, $\mathcal{B} = (B,\Sigma^{\mathcal{B}})$, $B = \{b_0,\ldots,b_l\}$, $\mathbf{b} = (B^{(1)},\ldots,B^{(n)})$ and $A \cap B = \emptyset$ such that $T(\mathfrak{A}) = S$ and $T(\mathfrak{B}) = T$. Let us also suppose that $a_0 \le \ldots \le a_k$ and $b_0 \le \ldots \le b_l$ hold on the state sets $A$ and $B$, respectively.

We construct a monotone $\mathfrak{C} = (\mathcal{C}, c_0, \mathbf{c})$ that recognizes $T \cdot_{x_i} S$ as follows. Let $\mathcal{C} = (C, \Sigma^{\mathcal{C}})$, $C = A \cup B$, $c_0 = a_0$ and $\mathbf{c} = (C^{(1)},\ldots,C^{(n)})$ hold, where $\mathbf{c}$ is defined as follows:

$$C^{(j)} = \begin{cases} A^{(j)} \cup B^{(j)} \cup A^{(i)}, & \text{if } x_j \in T,\ j \ne i \\ A^{(j)} \cup B^{(j)}, & \text{if } x_j \notin T,\ j \ne i \\ B^{(j)} \cup A^{(i)}, & \text{if } x_j \in T,\ j = i \\ B^{(j)}, & \text{if } x_j \notin T,\ j = i \end{cases}$$

It remains to represent the elements of $\Sigma$ in $\mathcal{C}$. For $\sigma \in \Sigma$ and $c \in C$ let

$$\sigma^{\mathcal{C}}(c) = \begin{cases} \sigma^{\mathcal{B}}(c), & \text{if } c \in B \\ \sigma^{\mathcal{B}}(b_0), & \text{if } c \in A^{(i)},\ \sigma \in root(T) \\ \sigma^{\mathcal{A}}(c), & \text{else} \end{cases}$$

The construction of $\mathfrak{C}$ relies on the condition $\Sigma_{S,x_i} \cap root(T) = \emptyset$. It allows us to determine at every step during the processing of a tree in $\mathfrak{C}$ whether the next input symbol is evaluated in $\mathfrak{A}$ or in $\mathfrak{B}$. Once we reach a state $a \in A^{(i)}$, the symbols from $root(T)$ will lead us to a state $b \in B$, from which we can continue the processing in $\mathfrak{B}$. If the input symbol applied in the state $a$ is from $\Sigma \setminus root(T)$, then we process it according to $\mathfrak{A}$. Therefore, it can be shown by a straightforward computation that $\mathfrak{C}$ recognizes $T \cdot_{x_i} S$, and $\mathfrak{C}$ is monotone under the linear ordering $a_0 \leq \ldots \leq a_k \leq b_0 \leq \ldots \leq b_l$, which means that $T \cdot_{x_i} S$ is monotone. □

**Corollary 19.** *Let $S, T \subseteq T_\Sigma(X_n)$ be monotone DR-languages, $x_i \in X_n$. If $\Sigma_S \cap root(T) = \emptyset$, then $T \cdot_{x_i} S$ is monotone.*

*Proof.* The conditions of Theorem 18 hold because $\Sigma_{S,x_i} \subseteq \Sigma_S$. □

The conversion of Theorem 18 does not hold as the counter example below shows.

**Example 20.** Let $T$ and $S$ stand for the DR-languages $\{\sigma(z,z)\}$ and $\{\sigma(x,z), \sigma(\sigma(z,z),z)\}$, respectively. It is obvious that $S$ and $T$ are monotone and $T \cdot_x S = \{\sigma(\sigma(z,z),\ z)\}$ is also monotone. However, $\Sigma_{S,x} \cap root(T) = \{\sigma\} \neq \emptyset$.

Let $x \in X_n$. A tree language $T$ is called *$x$-homogeneous* if there exists no $t \in T$ for which there are $u, v \in g_x(t)$, $w \in \hat{\Sigma}^*$ and $z \in X_n$ such that $uw \in g_z(T)$ and $vw \notin g_z(T)$.

The condition under which the class of monotone DR-languages is closed under $x$-iteration can be restricted by the following lemmas.

**Lemma 21.** *Let $T \subseteq T_\Sigma(X_n)$ be a DR-language, $x \in X_n$, and let $T^{*,x}$ be deterministic. If $T$ is not $x$-homogeneous, then $T^{*,x}$ is not monotone.*

*Proof.* Let us suppose that the conditions of the lemma hold. It means that there is a tree $t \in T$ for which there are $u, v \in g_x(t)$ with $u \neq v$, and there are $w \in \hat{\Sigma}^*$, $z \in X_n$ such that $uw \in g_z(T)$ and $vw \notin g_z(T)$. Moreover, let us assume that $\mathfrak{A}$ is a reduced monotone DR $\Sigma X_n$-recognizer which recognizes $T^{*,x}$. Let $a_i = a_0 u$ and $a_j = a_0 v$. Since $uw \in g_z(T)$ and $vw \notin g_z(T)$, we get that $a_i \neq a_j$. It is obvious that $a_i, a_j \in \alpha(x)$, hence $T(\mathfrak{A}, a_i) = T^{*,x}$ and $T(\mathfrak{A}, a_j) = T^{*,x}$. Using the fact that $\mathfrak{A}$ is reduced, $T(\mathfrak{A}, a_i) = T(\mathfrak{A}, a_j)$ implies that $a_i = a_j$, which is a contradiction. Therefore, $T^{*,x}$ is not monotone. □

**Lemma 22.** *Let $T \subseteq T_\Sigma(X_n)$ be a DR-language, $x \in X_n$, and let $T^{*,x}$ be deterministic. If $ih_x(T^{*,x}) > 1$, then $T^{*,x}$ is not monotone.*

*Proof.* Let us suppose that $T$ is a DR-language for which $T^{*,x}$ is deterministic and $ih_x(T^{*,x}) > 1$. Let the regular $\Sigma X_n$-expression $\zeta$ represent $T$. By the definition of $ih_x$, there is a reduced regular $\Sigma X_n$-expression $\eta$ for which $T(\eta) = T^{*,x}$, $ih_x(\eta) > 1$ and $\eta$ is in form $(\zeta)^{*,x}$. Using Lemma 8 we get that $T(\eta)$ is not monotone, therefore $T^{*,x}$ is not monotone, too. □

Now we give a condition by which the $x$-iteration of a monotone DR-language is also monotone.

**Theorem 23.** *Let $T \subseteq T_\Sigma(X_n)$ be a monotone DR-language, $x_i \in X_n$, and let $T^{*,x_i}$ be deterministic. If $T$ is $x_i$-homogeneous, $ih_{x_i}(T^{*,x_i}) \leq 1$ and $\Sigma_{T,x_i} \cap root(T) = \emptyset$, then $T^{*,x_i}$ is monotone.*

*Proof.* Let us suppose that the conditions of the theorem hold. Let $\mathfrak{A}$ be a reduced DR $\Sigma X_n$-recognizer for which $T(\mathfrak{A}) = T$, and where $\mathfrak{A} = (\mathcal{A}, a_0, \mathbf{a})$, $\mathcal{A} = (A, \Sigma^{\mathcal{A}})$, $A = \{a_0, \ldots, a_k\}$, $\mathbf{a} = (A^{(1)}, \ldots, A^{(n)})$. Let us also assume that $\mathfrak{A}$ is monotone under the linear ordering $a_0 \leq \ldots \leq a_k$.

We construct the monotone DR $\Sigma X_n$-recognizer $\mathfrak{B} = (\mathcal{B}, b_0, \mathbf{b})$ with $\mathcal{B} = (B, \Sigma^{\mathcal{B}})$ which recognizes $T^{*,x_i}$. Let us define the state set $B$ as $A \cup \{b_0\}$, where $b_0$ is a new state. The final state vector $\mathbf{b}$ is

$$(B^{(1)}, \ldots, B^{(i-1)}, \{a_0, b_0\}, B^{(i+1)}, \ldots, B^{(n)}),$$

where the components are defined by two steps in the following order:

(1) For all $j \in \{1, \ldots, n\} \setminus \{i\}$, $B^{(j)} := \begin{cases} A^{(j)} \cup \{b_0\}, & \text{if } a_0 \in A^{(j)} \\ A^{(j)}, & \text{else,} \end{cases}$

(2) For all $a \in A^{(i)}$ and $j \in \{1, \ldots, i-1, i+1, \ldots, n\}$ if $a \in A^{(j)}$, then $B^{(j)} := B^{(j)} \cup \{a_0\}$.

The definition of $\Sigma^{\mathcal{B}}$ is given by four steps in the following order:

(3) For all $\sigma \in root(T)$ and $a' \in A^{(i)}$

$$\sigma^{\mathcal{B}}(a_0) := \begin{cases} (\ldots, a_0, \ldots), & \text{if } \sigma^{\mathcal{A}}(a_0) = (\ldots, a', \ldots) \\ \sigma^{\mathcal{A}}(a_0), & \text{else,} \end{cases}$$

(4) For all $\sigma \in \Sigma \setminus root(T)$

$$\sigma^{\mathcal{B}}(a_0) := \begin{cases} \sigma^{\mathcal{A}}(a'), & \text{if } A^{(i)} \neq \emptyset, \ (a' \in A^{(i)} \text{ is arbitrarily chosen}) \\ \sigma^{\mathcal{A}}(a_0), & \text{if } A^{(i)} = \emptyset, \end{cases}$$

(5) For all $\sigma \in root(T)$ $\sigma^{\mathcal{B}}(b_0) := \sigma^{\mathcal{B}}(a_0)$,

(6) For all $\sigma \in \Sigma$ and $a \in A \setminus \{a_0\}$ $\sigma^{\mathcal{B}}(a) := \sigma^{\mathcal{A}}(a)$.

The construction of $\mathfrak{B}$ relies on the condition $\Sigma_{T,x_i} \cap root(T) = \emptyset$. It guarantees us that in every state $a \in \alpha(x_i)$ for any input symbol $\sigma$ we can determine whether to continue an already started processing of a tree, or to start a process from the root of a tree from $T$. In all the other cases $\mathfrak{B}$ is acting as $\mathfrak{A}$ did. The $x_i$-homogeneous property of $T$ and the inequality $ih_{x_i}(T^{*,x_i}) \leq 1$ ensure us that one state is enough to iterate the $x_i$-paths of $T$, which is the basic idea of any iteration related automata construction. Therefore, it can be shown by a straightforward computation that $T(\mathfrak{B}) = T^{*,x_i}$, and $\mathfrak{B}$ is monotone under the linear ordering $b_0 \leq a_0 \leq \ldots \leq a_k$, which means that $T^{*,x_i}$ is monotone. □

The following lemma is obvious.

**Lemma 24.** *For any fixed variable $x \in X_n$ the $x$-product of tree languages is associative, that is, for any tree languages $S$, $R$ and $T$ the equality $T \cdot_x (R \cdot_x S) = (T \cdot_x R) \cdot_x S$ holds.*

A tree language $\eta = \eta_k \cdot_{\xi_k} \ldots \cdot_{\xi_1} \eta_0$ is called *$R$-chain language*, if every $\eta_i$ is in form $(T_i) \cdot_{\xi_i} (S_i)^{*,\xi_i}$ $(i = 0, \ldots, k)$, where $S_i$ and $T_i$ are finite DR-languages, for which $S_i$ is $\xi_i$-homogeneous, $ih_{\xi_i}(S_i) \leq 1$, $root(S_i) \cap \Sigma_{S_i,\xi_i} = \emptyset$ and $root(T_i) \cap (root(S_i) \cup \Sigma_{S_i,\xi_i}) = \emptyset$. Moreover, let us denote the language $\eta_{i-1} \cdot_{\xi_{i-1}} \cdots \cdot_{\xi_1} \eta_0$ by $\zeta_i$. The $\eta = \eta_k \cdot_{\xi_k} \ldots \cdot_{\xi_1} \eta_0$ $R$-chain language is called *generalized*, if $root(T(\eta_i)) \cap \Sigma_{T(\zeta_i),\xi_i} = \emptyset$ holds for every $i = 1, \ldots, k$.

**Theorem 25.** *Let $T$ be a DR-language. $T$ is monotone iff it can be given as a generalized $R$-chain language.*

*Proof.* Let us suppose that $T$ is a monotone DR-language. Let $\mathfrak{A}$ be the monotone DR-recognizer for which $T(\mathfrak{A}) = T$. Constructing the regular expression $\eta_{\mathfrak{A}}$ belonging to $\mathfrak{A}$ we get a generalized $R$-chain language for which $T = T(\eta_{\mathfrak{A}})$.

Conversely, let us take a generalized $R$-chain language $\eta = \eta_k \cdot_{\xi_k} \ldots \cdot_{\xi_1} \eta_0$ which represents $T$. From Lemma 6, Theorem 18, and Theorem 23 we directly obtain that every $T(\eta_i)$ is monotone $(i = 0, \ldots, k)$. Using Lemma 24 and Theorem 18 we directly get that $T(\eta)$ is monotone. □

# 9 Conclusion

As we showed above, the monotone DR-languages can be characterized by means of generalized $R$-chain languages. We gave several conditions by which some particular operations preserve monotonicity, but we did not state conditions by which the class of DR-languages is closed under the operations of $x$-product, $x$-iteration and union. However, it seems possible to give appropriate conditions for each operation mentioned above.

# 10 Acknowledgement

The author is extremely grateful to Professor Ferenc Gécseg for his helpful comments and valuable suggestions. The author is also thankful to the anonymous reviewer whose remarks considerably improved the quality and style of this paper.

# References

[1] Courcelle, B.: A representation of trees by languages I, *Theoretical Computer Science*, **6** (1978), 255–279.

[2] Gécseg, F.: On some classes of tree automata and tree languages, *Annales Academiæ Scientiarum Fennicæ, Mathematica*, **25** (2000), 325–336.

[3] Gécseg, F. and Gyurica, Gy.: On the closedness of nilpotent DR tree languages under Boolean operations, *Acta Cybernetica,* **17** (2006), 449–457.

[4] Gécseg, F. and Imreh, B.: On monotone automata and monotone languages, *Journal of Automata, Languages, and Combinatorics,* **7** (2002), 71–82.

[5] Gécseg, F. and Steinby, M.: Minimal ascending tree automata, *Acta Cybernetica,* **4** (1978), 37–44.

[6] Gécseg, F. and Steinby, M.: Minimal Recognizers and Syntactic Monoids of DR Tree Languages, In *Words, Semigroups, & Transductions*, World Scientifics (2001), 155–167.

[7] Gécseg, F. and Steinby, M.: *Tree Automata*, Akadémiai Kiadó, Budapest 1984.

[8] Jurvanen, E.: The Boolean closure of DR-recognizable tree languages, *Acta Cybernetica,* **10** (1992), 255–272.

[9] Virágh, J.: Deterministic ascending tree automata I, *Acta Cybernetica*, **5** (1980), 33-42.

*Received December, 2005*

Acta Cybernetica 18 (2007) 135–153.

# Self-Regulating Finite Automata

Alexander Meduna* and Tomáš Masopust*

**Abstract**

This paper introduces and discusses *self-regulating finite automata*. In essence, these automata regulate the use of their rules by a sequence of rules applied during previous moves. A special attention is paid to *turns* defined as moves during which a self-regulating finite automaton starts a new self-regulating sequence of moves. Based on the number of turns, the present paper establishes two infinite hierarchies of language families resulting from two variants of these automata. In addition, it demonstrates that these hierarchies coincide with the hierarchies resulting from parallel right linear grammars and right linear simple matrix grammars, so the self-regulating finite automata can be viewed as the automaton counterparts to these grammars. Finally, this paper compares both infinite hierarchies. In addition, as an open problem area, it suggests the discussion of self-regulating pushdown automata and points out that they give rise to no infinite hierarchy analogical to the achieved hierarchies resulting from the self-regulating finite automata.

**Keywords:** regulated automata, self-regulation, infinite hierarchies of language families, parallel right linear grammars, right linear simple matrix grammars

## 1 Introduction

Over its history, automata theory has modified and restricted classical automata in many ways (see [3, 5, 6, 7, 8, 16, 22, 24, 26]). Recently, regulated automata have been introduced and studied in [17, 18]. In essence, these automata regulate the use of their rules according to which they make moves by control languages. In this paper, we continue with this topic by defining and investigating *self-regulating finite automata*. Instead of prescribed control languages, however, the self-regulating finite automata restrict the selection of a rule according to which the current move is made by a rule according to which a previous move was made.

To give a more precise insight into self-regulating automata, consider a finite automaton, $M$, with a finite binary relation, $R$, over $M$'s rules. Furthermore, suppose that $M$ makes a sequence of moves, $\rho$, that leads to the acceptance of a

*Department of Information Systems, Faculty of Information Technology, Brno University of Technology, Božetěchova 2, Brno 61266, Czech Republic E-mail: meduna@fit.vutbr.cz, masopust@fit.vutbr.cz

word, so $\rho$ can be expressed as a concatenation of $n+1$ consecutive subsequences, $\rho = \rho_0\rho_1 \dots \rho_n$, $|\rho_i| = |\rho_j|$, $0 \leq i, j \leq n$, in which $r_i^j$ denote the rule according to which the $i$th move in $\rho_j$ is made, for all $0 \leq j \leq n$ and $1 \leq i \leq |\rho_j|$ (as usual, $|\rho_j|$ denotes the length of $\rho_j$). If for all $0 \leq j < n$, $(r_1^j, r_1^{j+1}) \in R$, then $M$ represents an *$n$-turn first-move self-regulating finite automaton with respect to $R$*. If for all $0 \leq j < n$ and all $1 \leq i \leq |\rho_i|$, $(r_i^j, r_i^{j+1}) \in R$, then $M$ represents an *$n$-turn all-move self-regulating finite automaton with respect to $R$*.

Based on the number of turns, we establish two infinite hierarchies of language families that lie between the families of regular and context-sensitive languages. First, we demonstrate that $n$-turn first-move self-regulating finite automata give rise to an infinite hierarchy of language families coinciding with the hierarchy resulting from $(n+1)$-parallel right linear grammars (see [20, 21, 27, 28]). Recall that $n$-parallel right linear grammars generate a proper language subfamily of the language family generated by $(n+1)$-parallel right linear grammars (see Theorem 5 in [21]). As a result, $n$-turn first-move self-regulating finite automata accept a proper language subfamily of the language family accepted by $(n+1)$-turn first-move self-regulating finite automata, for all $n \geq 0$. Similarly, we prove that $n$-turn all-move self-regulating finite automata give rise to an infinite hierarchy of language families coinciding with the hierarchy resulting from $(n+1)$-right linear simple matrix grammars (see [4, 10, 28]). As $n$-right linear simple matrix grammars generate a proper subfamily of the language family generated by $(n+1)$-right linear simple matrix grammars (see Theorem 1.5.4 in [4]), $n$-turn all-move self-regulating finite automata accept a proper language subfamily of the language family accepted by $(n+1)$-turn all-move self-regulating finite automata. Furthermore, since the families of right linear simple matrix languages coincide with the language families accepted by multitape nonwriting automata (see [5]) and by finite-turn checking automata (see [24]), the all-move self-regulating finite automata characterize these families, too. Finally, we summarize the results about both infinite hierarchies.

In the conclusion of this paper, as an open problem area, we suggest the discussion of *self-regulating pushdown automata*. Regarding self-regulating all-move pushdown automata, we prove that they do not give rise to any infinite hierarchy analogical to the achieved hierarchies resulting from the self-regulating finite automata. Indeed, zero-turn all-move self-regulating pushdown automata define the family of context-free languages while one-turn all-move self-regulating pushdown automata define the family of recursively enumerable languages. On the other hand, as far as self-regulating first-move pushdown automata are concerned, the question whether they define an infinite hierarchy or not is open.

## 2 Preliminaries

We assume that the reader is familiar with the theory of automata and formal languages (see [1, 2, 9, 11, 12, 13, 15, 19, 25]). For a set $Q$, $|Q|$ denotes the cardinality of $Q$. $\mathbb{N} = \{1, 2, 3, \dots\}$ denotes the set of all natural numbers. For an alphabet $V$, $V^*$ represents the free monoid generated by $V$ under the operation of

concatenation. The identity of $V^*$ is denoted by $\varepsilon$. Set $V^+ = V^* - \{\varepsilon\}$; algebraically, $V^+$ is thus the free semigroup generated by $V$ under the operation of concatenation. For $w \in V^*$, $|w|$ denotes the length of $w$. Let $w \in V^*$; then, $alph(w) = \{a \in V : a$ appears in $w\}$. For every $L \subseteq V^*$, $alph(L) = \bigcup_{w \in L} alph(w)$.

A *finite automaton*, $M$, is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$, where $Q$ is a finite set of states, $\Sigma$ is a finite input alphabet, $\delta$ is a finite set of rules of the form $qw \to p$, $q, p \in Q$, $w \in \Sigma^*$, $q_0 \in Q$ is an initial state, and $F$ is a set of final states. Let $\Psi$ be an alphabet of *rule labels* such that $|\Psi| = |\delta|$, and $\psi$ be a bijection from $\delta$ to $\Psi$. For simplicity, to express that $\psi$ maps a rule $qw \to p \in \delta$ to $r$, where $r \in \Psi$, we write $r.qw \to p \in \delta$; in other words, $r.qw \to p$ means $\psi(qw \to p) = r$. A *configuration* of $M$ is any word from $Q\Sigma^*$. For any configuration $qwy$, where $y \in \Sigma^*$, $q \in Q$, and any $r.qw \to p \in \delta$, $M$ makes a move from configuration $qwy$ to configuration $py$ according to $r$, written as $qwy \Rightarrow py\,[r]$. Let $\chi$ be any configuration of $M$. $M$ makes zero moves from $\chi$ to $\chi$ according to $\varepsilon$, written as $\chi \Rightarrow^0 \chi\,[\varepsilon]$. Let there exist a sequence of configurations $\chi_0, \chi_1, \ldots, \chi_n$, for some $n \geq 1$, such that $\chi_{i-1} \Rightarrow \chi_i\,[r_i]$, where $r_i \in \Psi$, $i = 1, \ldots, n$. Then, $M$ makes $n$ moves from $\chi_0$ to $\chi_n$ according to $r_1, \ldots, r_n$, symbolically written as $\chi_0 \Rightarrow^n \chi_n\,[r_1 \ldots r_n]$. We write $\varphi \Rightarrow^* \kappa\,[\mu]$ if $\varphi \Rightarrow^n \kappa\,[\mu]$ for some $n \geq 0$. If $w \in \Sigma^*$ and $q_0 w \Rightarrow^* f\,[\mu]$, for $f \in F$, then $w$ *is accepted by* $M$ and $q_0 w \Rightarrow^* f\,[\mu]$ is an *acceptance* of $w$ in $M$. The *language* of $M$ is defined as $\mathcal{L}(M) = \{w \in \Sigma^* : q_0 w \Rightarrow^* f\,[\mu]$ is an acceptance of $w\}$.

For $n \geq 1$, an *$n$-parallel right linear grammar*, $n$-PRLG, is an $(n+3)$-tuple $G = (N_1, \ldots, N_n, T, S, P)$, where $N_i$, $1 \leq i \leq n$, are mutually disjoint nonterminal alphabets, $T$ is a terminal alphabet, $S \notin N$ is an initial symbol, $N = N_1 \cup \cdots \cup N_n$, and $P$ is a finite set of rules that contains three kinds of rules:

1. $S \to X_1 \ldots X_n$, $\quad X_i \in N_i$, $1 \leq i \leq n$;
2. $X \to wY$, $\quad X, Y \in N_i$ for some $i$, $1 \leq i \leq n$, $w \in T^*$;
3. $X \to w$, $\quad X \in N$, $w \in T^*$.

For $x, y \in (N \cup T \cup \{S\})^*$, $x \Rightarrow y$ if and only if

1. either $x = S$ and $S \to y \in P$,
2. or $x = y_1 X_1 \ldots y_n X_n$, $y = y_1 x_1 \ldots y_n x_n$, where $y_i \in T^*$, $x_i \in T^* N \cup T^*$, $X_i \in N_i$, and $X_i \to x_i \in P$, $1 \leq i \leq n$.

If $x, y \in (N \cup T \cup \{S\})^*$ and $l > 0$, then $x \Rightarrow^l y$ if and only if there exists a sequence $x_0 \Rightarrow x_1 \Rightarrow \cdots \Rightarrow x_l$, $x_0 = x$, $x_l = y$. Then, we say $x \Rightarrow^+ y$ if and only if there exists $l > 0$ such that $x \Rightarrow^l y$, and $x \Rightarrow^* y$ if and only if $x = y$ or $x \Rightarrow^+ y$. The language generated by an $n$-PRLG, $G$, is defined as $\mathcal{L}(G) = \{w \in T^* : S \Rightarrow^+ w\}$. Language $L \subseteq T^*$ is an *$n$-parallel right linear language*, $n$-PRLL, if there is an $n$-PRLG, $G$, such that $L = \mathcal{L}(G)$. The family of $n$-PRLLs is denoted by $R_n$.

For $n \geq 1$, an *$n$-right linear simple matrix grammar*, $n$-RLSMG, is an $(n+3)$-tuple $G = (N_1, \ldots, N_n, T, S, P)$, where $N_i$, $1 \leq i \leq n$, are mutually disjoint nonterminal alphabets, $T$ is a terminal alphabet, $S \notin N$ is an initial symbol, $N = N_1 \cup \cdots \cup N_n$, and $P$ is a finite set of matrix rules. A matrix rule can be in one of the following three forms:

1. $[S \to X_1 \dots X_n]$, $X_i \in N_i$, $1 \leq i \leq n$;
2. $[X_1 \to w_1 Y_1, \dots, X_n \to w_n Y_n]$, $w_i \in T^*$, $X_i, Y_i \in N_i$, $1 \leq i \leq n$;
3. $[X_1 \to w_1, \dots, X_n \to w_n]$, $X_i \in N_i$, $w_i \in T^*$, $1 \leq i \leq n$.

Let $m$ be a matrix, then $m[i]$ denotes the $i$th rule of $m$. For $x, y \in (N \cup T \cup \{S\})^*$, $x \Rightarrow y$ if and only if

1. either $x = S$ and $[S \to y] \in P$,
2. or $x = y_1 X_1 \dots y_n X_n$, $y = y_1 x_1 \dots y_n x_n$, where $y_i \in T^*$, $x_i \in T^* N \cup T^*$, $X_i \in N_i$, $1 \leq i \leq n$, and $[X_1 \to x_1, \dots, X_n \to x_n] \in P$.

We define $x \Rightarrow^+ y$ and $x \Rightarrow^* y$ as above. The language generated by an $n$-RLSMG, $G$, is defined as $\mathcal{L}(G) = \{w \in T^* : S \Rightarrow^+ w\}$. Language $L \subseteq T^*$ is an *$n$-right linear simple matrix language*, $n$-RLSML, if there is an $n$-RLSMG, $G$, such that $L = \mathcal{L}(G)$. The family of $n$-RLSMLs is denoted by $R_{[n]}$.

Let $G = (N_1, \dots, N_n, T, S, P)$ be an $n$-PRLG, for some $n \geq 1$, and $1 \leq i \leq n$. By the *$i$th component of $G$* we understand a 1-PRLG $G = (N_i, T, S', P')$, where $P'$ contains rules of the following forms:

1. $S' \to X_i$ if $S \to X_1 \dots X_n \in P$, $X_i \in N_i$;
2. $X \to wY$ if $X \to wY \in P$ and $X, Y \in N_i$;
3. $X \to w$ if $X \to w \in P$ and $X \in N_i$.

The $i$th component of an $n$-RLSMG is defined analogously.

Finally, let $REG$, $CF$, and $CS$ denote the families of regular, context-free, and context-sensitive languages, respectively.

# 3 Definitions and Examples

In this section, we define and illustrate $n$-turn first-move self-regulating finite automata and $n$-turn all-move self-regulating finite automata.

**Definition 1.** *A* self-regulating finite automaton, *SFA*, *$M$, is a septuple*

$$M = (Q, \Sigma, \delta, q_0, q_t, F, R),$$

*where*

1. *$(Q, \Sigma, \delta, q_0, F)$ is a finite automaton,*
2. *$q_t \in Q$ is a* turn state, *and*
3. *$R \subseteq \Psi \times \Psi$ is a finite relation on the alphabet of rule labels.*

In this paper, we consider two ways of self-regulation—first-move and all-move. According to these two types of self-regulation, two types of $n$-turn self-regulating finite automata are defined.

**Definition 2.** *Let $n \geq 0$ and $M = (Q, \Sigma, \delta, q_0, q_t, F, R)$ be a self-regulating finite automaton. $M$ is said to be an* $n$-turn first-move self-regulating finite automaton, *$n$-first-SFA, if $M$ accepts $w$ in the following way. There is an acceptance of the form $q_0 w \Rightarrow^* f\,[\mu]$ such that*

$$\mu = r_1^0 \ldots r_k^0 r_1^1 \ldots r_k^1 \ldots r_1^n \ldots r_k^n,$$

*where $k \in \mathbb{N}$, $r_k^0$ is the first rule of the form $qx \rightarrow q_t$, for some $q \in Q$, $x \in \Sigma^*$, and*

$$(r_1^j, r_1^{j+1}) \in R$$

*for all $0 \leq j < n$.*

*The family of languages accepted by $n$-first-SFAs is denoted by $W_n$.*

**Example 3.** Consider a 1-turn first-move self-regulating finite automaton, $M = (\{s,t,f\}, \{a,b\}, \delta, s, t, \{f\}, \{(1,3)\})$, with $\delta$ containing rules $1.sa \rightarrow s$, $2.sa \rightarrow t$, $3.tb \rightarrow f$, and $4.fb \rightarrow f$ (see Fig. 1).

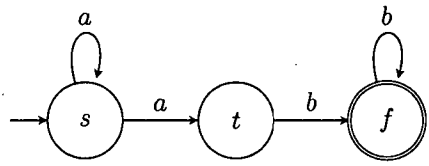


Figure 1: 1-turn first-move self-regulating finite automaton $M$.

With *aabb*, $M$ makes

$$saabb \Rightarrow sabb\,[1] \Rightarrow tbb\,[2] \Rightarrow fb\,[3] \Rightarrow f\,[4].$$

In brief, $saabb \Rightarrow^* f\,[1234]$. Observe that $\mathcal{L}(M) = \{a^n b^n : n \geq 1\}$, which belongs to $CF - REG$.

**Definition 4.** *Let $n \geq 0$ and $M = (Q, \Sigma, \delta, q_0, q_t, F, R)$ be a self-regulating finite automaton. $M$ is said to be an* $n$-turn all-move self-regulating finite automaton, *$n$-all-SFA, if $M$ accepts $w$ in the following way. There is an acceptance $q_0 w \Rightarrow^* f\,[\mu]$ such that*

$$\mu = r_1^0 \ldots r_k^0 r_1^1 \ldots r_k^1 \ldots r_1^n \ldots r_k^n,$$

*where $k \in \mathbb{N}$, $r_k^0$ is the first rule of the form $qx \rightarrow q_t$, for some $q \in Q$, $x \in \Sigma^*$, and*

$$(r_i^j, r_i^{j+1}) \in R$$

*for all $1 \leq i \leq k$, $0 \leq j < n$.*

*The family of languages accepted by $n$-all-SFAs is denoted by $S_n$.*

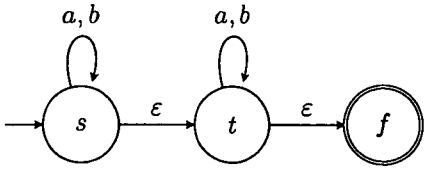


Figure 2: 1-turn all-move self-regulating finite automaton $M$.

**Example 5.** Consider a 1-turn all-move self-regulating finite automaton, $M = (\{s,t,f\}, \{a,b\}, \delta, s, t, \{f\}, \{(1,4),(2,5),(3,6)\})$, with $\delta$ containing rules $1.sa \to s$, $2.sb \to s$, $3.s \to t$, $4.ta \to t$, $5.tb \to t$, and $6.t \to f$ (see Fig. 2).
With $abab$, $M$ makes

$$sabab \Rightarrow sbab\,[1] \Rightarrow sab\,[2] \Rightarrow tab\,[3] \Rightarrow tb\,[4] \Rightarrow t\,[5] \Rightarrow f\,[6].$$

In brief, $sabab \Rightarrow^* f\,[123456]$. Observe that $\mathcal{L}(M) = \{ww : w \in \{a,b\}^*\}$, which belongs to $CS - CF$.

# 4 Results

We prove that the family of languages accepted by $n$-first-SFAs coincides with the family of languages generated by $(n+1)$-PRLGs. Furthermore, we demonstrate that the family of languages accepted by $n$-all-SFAs coincides with the family of languages generated by $n$-RLSMGs.

## 4.1 $n$-Turn First-Move Self-Regulating Finite Automata

Section 4.1 establishes the identity between the family of languages accepted by $n$-first-SFAs and the family of languages generated by $(n+1)$-PRLGs. To do so, we need the following form of parallel right linear grammars.

**Lemma 6.** *For every $n$-PRLG $G = (N_1, \ldots, N_n, T, S, P)$, there is an equivalent $n$-PRLG $G' = (N'_1, \ldots, N'_n, T, S, P')$ that satisfies:*

1. *if $S \to X_1 \ldots X_n \in P'$, then $X_i$ does not occur on the right-hand side of any rule, for $1 \leq i \leq n$;*
2. *if $S \to \alpha$, $S \to \beta \in P'$ and $\alpha \neq \beta$, then $alph(\alpha) \cap alph(\beta) = \emptyset$.*

*Proof.* If $G$ does not satisfy conditions from the lemma, then we will construct a new $n$-PRLG $G' = (N'_1, \ldots, N'_n, T, S, P')$, where $P'$ contains all rules of the form $X \to \beta \in P$, $X \neq S$, and $N_j \subseteq N'_j$, $1 \leq j \leq n$. For each rule $S \to X_1 \ldots X_n \in P$, we add new nonterminals $Y_j \notin N'_j$ into $N'_j$, and rules include $S \to Y_1 \ldots Y_n$ and $Y_j \to X_j$ in $P'$, $1 \leq j \leq n$. Clearly,

$$S \Rightarrow_G X_1 \ldots X_n \text{ if and only if } S \Rightarrow_{G'} Y_1 \ldots Y_n \Rightarrow X_1 \ldots X_n.$$

Thus, $\mathcal{L}(G) = \mathcal{L}(G')$. □

**Lemma 7.** *Let $G$ be an $n$-PRLG. Then, there is an $(n-1)$-first-SFA, $M$, such that $\mathcal{L}(G) = \mathcal{L}(M)$.*

*Proof.* Informally, $M$ is divided into $n$ parts (see Fig. 3). The $i$th part represents a finite automaton accepting the language of $G$'s $i$th component, and $R$ also connects the $i$th part to the $(i+1)$st part as depicted in Fig. 3.

Formally, without loss of generality, we assume $G = (N_1, \ldots, N_n, T, S, P)$ to be in the form from Lemma 6. We construct an $(n-1)$-first-SFA $M = (Q, T, \delta, q_0, q_t, F, R)$, where $Q = \{q_0, \ldots, q_n\} \cup N$, $N = N_1 \cup \cdots \cup N_n$, $\{q_0, q_1, \ldots, q_n\} \cap N = \emptyset$, $F = \{q_n\}$, $\delta = \{q_i \to X_{i+1} : S \to X_1 \ldots X_n \in P, 0 \leq i < n\} \cup \{Xw \to Y : X \to wY \in P\} \cup \{Xw \to q_i : X \to w \in P, w \in T^*, X \in N_i, i \in \{1, \ldots, n\}\}$, $q_t = q_1$, $\Psi = \delta$ with the identity map, and $R = \{(q_i \to X_{i+1}, q_{i+1} \to X_{i+2}) : S \to X_1 \ldots X_n \in P, 0 \leq i \leq n-2\}$.

Next, we prove $\mathcal{L}(G) = \mathcal{L}(M)$. To prove $\mathcal{L}(G) \subseteq \mathcal{L}(M)$, consider a derivation of $w$ in $G$ and construct an acceptance of $w$ in $M$ depicted in Fig. 3. This figure clearly

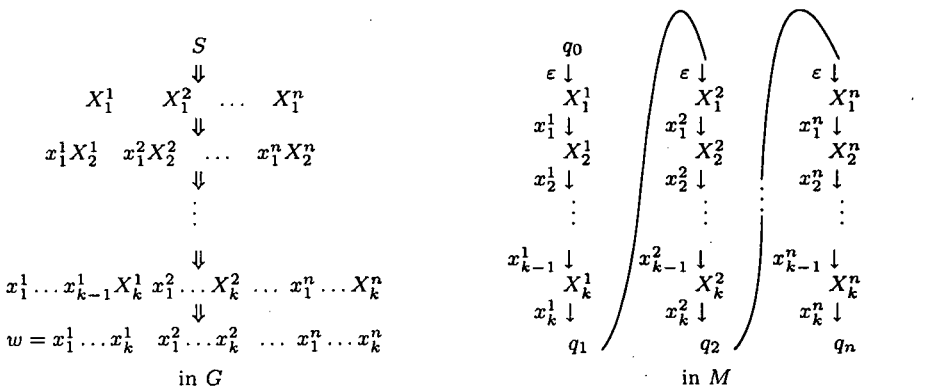


Figure 3: A derivation of $w$ in $G$ and the corresponding acceptance of $w$ in $M$.

demonstrates the fundamental idea behind this part of the proof; its complete and rigorous version is lengthy and left to the reader. Thus, for each derivation $S \Rightarrow^* w$, $w \in T^*$, there is an acceptance of $w$ in $M$.

To prove $\mathcal{L}(M) \subseteq \mathcal{L}(G)$, let $w \in \mathcal{L}(M)$. Consider an acceptance of $w$ in $M$. Observe that the acceptance is of the form depicted on the right-hand side of Fig. 3. It means that the number of steps $M$ made from $q_{i-1}$ to $q_i$ is the same as from $q_i$ to $q_{i+1}$ since the only rule in the relation with $q_{i-1} \to X_1^i$ is the rule $q_i \to X_1^{i+1}$. Moreover, $M$ can never come back to a state corresponding to a previous component. (By a component of $M$, we mean the finite automaton $M_i = (Q, \Sigma, \delta, q_{i-1}, \{q_i\})$, for $1 \leq i \leq n$.) Now, construct a derivation of $w$ in $G$. By Lemma 6, we have $|\{X : (q_i \to X_1^{i+1}, q_{i+1} \to X) \in R\}| = 1$, for all $0 \leq i < n-1$. Thus, $S \to X_1^1 X_1^2 \ldots X_1^n \in P$. Moreover, if $X_j^i x_j^i \to X_{j+1}^i$, we apply

$X_j^i \to x_j^i X_{j+1}^i \in P$, and if $X_k^i x_k^i \to q_i$, we apply $X_k^i \to x_k^i \in P$, $1 \leq i \leq n$, $1 \leq j < k$.

Hence, Lemma 7 holds. □

**Lemma 8.** *Let $M$ be an $n$-first-SFA. There is an $(n+1)$-PRLG, $G$, such that $\mathcal{L}(G) = \mathcal{L}(M)$.*

*Proof.* Let $M = (Q, \Sigma, \delta, q_0, q_t, F, R)$. Consider $G = (N_0, \ldots, N_n, \Sigma, S, P)$, where $N_i = (Q\Sigma^l \times Q \times \{i\} \times Q) \cup (Q \times \{i\} \times Q)$, $l = \max\{|w| : qw \to p \in \delta\}$, $0 \leq i \leq n$, and

$$
\begin{aligned}
P = \{ & S \to [q_0x_0, q^0, 0, q_t][q_tx_1, q^1, 1, q_{i_1}][q_{i_1}x_2, q^2, 2, q_{i_2}] \ldots [q_{i_{n-1}}x_n, q^n, n, q_{i_n}] : \\
& r_0.q_0x_0 \to q^0,\ r_1.q_tx_1 \to q^1,\ r_2.q_{i_1}x_2 \to q^2, \ldots,\ r_n.q_{i_{n-1}}x_n \to q^n \in \delta, \\
& (r_0, r_1), (r_1, r_2), \ldots, (r_{n-1}, r_n) \in R,\ q_{i_n} \in F\} \cup \\
& \{[px, q, i, r] \to x[q, i, r]\} \cup \\
& \{[q, i, q] \to \varepsilon : q \in Q\} \cup \\
& \{[q, i, p] \to w[q', i, p] : qw \to q' \in \delta\}.
\end{aligned}
$$

Next, we prove $\mathcal{L}(G) = \mathcal{L}(M)$. To prove $\mathcal{L}(G) \subseteq \mathcal{L}(M)$, observe that we make $n+1$ copies of $M$ and go through them similarly to Fig. 3. Consider a derivation of $w$ in $G$. Then, in greater detail, this derivation is of the form

$$
\begin{aligned}
S &\Rightarrow [q_0x_0^0, q_1^0, 0, q_t][q_tx_0^1, q_1^1, 1, q_{i_1}] \ldots [q_{i_{n-1}}x_0^n, q_1^n, n, q_{i_n}] \\
&\Rightarrow x_0^0[q_1^0, 0, q_t]x_0^1[q_1^1, 1, q_{i_1}] \ldots x_0^n[q_1^n, n, q_{i_n}] \\
&\Rightarrow x_0^0x_1^0[q_2^0, 0, q_t]x_0^1x_1^1[q_2^1, 1, q_{i_1}] \ldots x_0^nx_1^n[q_2^n, n, q_{i_n}] \qquad (1) \\
&\ \ \vdots \\
&\Rightarrow x_0^0x_1^0 \ldots x_k^0[q_t, 0, q_t]x_0^1x_1^1 \ldots x_k^1[q_{i_1}, 1, q_{i_1}] \ldots x_0^nx_1^n \ldots x_k^n[q_{i_n}, n, q_{i_n}] \\
&\Rightarrow x_0^0x_1^0 \ldots x_k^0x_0^1x_1^1 \ldots x_k^1 \ldots x_0^nx_1^n \ldots x_k^n
\end{aligned}
$$

and $r_0.q_0x_0^0 \to q_1^0$, $r_1.q_tx_0^1 \to q_1^1$, $r_2.q_{i_1}x_0^2 \to q_1^2, \ldots, r_n.q_{i_{n-1}}x_0^n \to q_1^n \in \delta$, $(r_0, r_1)$, $(r_1, r_2), \ldots, (r_{n-1}, r_n) \in R$, and $q_{i_n} \in F$.

Thus, the list of rules used in the acceptance of $w$ in $M$ is

$$
\begin{aligned}
\mu = \ & (q_0x_0^0 \to q_1^0)(q_1^0x_1^0 \to q_2^0) \ldots (q_k^0x_k^0 \to q_t) \\
& (q_tx_0^1 \to q_1^1)(q_1^1x_1^1 \to q_2^1) \ldots (q_k^1x_k^1 \to q_{i_1}) \\
& (q_{i_1}x_0^2 \to q_1^2)(q_1^2x_1^2 \to q_2^2) \ldots (q_k^2x_k^2 \to q_{i_2}) \qquad (2) \\
& \ \ \vdots \\
& (q_{i_{n-1}}x_0^n \to q_1^n)(q_1^nx_1^n \to q_2^n) \ldots (q_k^nx_k^n \to q_{i_n}).
\end{aligned}
$$

Now, we prove $\mathcal{L}(M) \subseteq \mathcal{L}(G)$. Informally, the acceptance is divided into $n+1$ parts of the same length. Grammar $G$ generates the $i$th part by the $i$th component and records the state from which the next component starts.

Let $\mu$ be a list of rules used in an acceptance of $w$ in $M$ of the form (2), where $w = x_0^0 x_1^0 \dots x_k^0 x_0^1 x_1^1 \dots x_k^1 \dots x_0^n x_1^n \dots x_k^n$. Then, the derivation of the form (1) is the corresponding derivation of $w$ in $G$ since $[q_j^i, i, p] \to x_j^i [q_{j+1}^i, i, p] \in P$ and $[q, i, q] \to \varepsilon$, for all $0 \leq i \leq n$, $1 \leq j < k$.

Hence, Lemma 8 holds. □

The first main result of this paper follows next.

**Theorem 9.** *For all $n \geq 0$, $W_n = R_{n+1}$.*

*Proof.* This proof follows from Lemma 7 and 8. □

**Corollary 10.** *The following statements hold true.*

1. $REG = W_0 \subset W_1 \subset W_2 \subset \cdots \subset CS$.
2. $W_1 \subset CF$.
3. $W_2 \not\subseteq CF$.
4. $CF \not\subseteq W_n$ *for any* $n \geq 0$.
5. *For all $n \geq 0$, $W_n$ is closed under union, finite substitution, homomorphism, intersection with a regular language, and right quotient with a regular language.*
6. *For all $n \geq 1$, $W_n$ is not closed under intersection and complement.*

*Proof.* Recall the following statements proved in [21]:

- $REG = R_1 \subset R_2 \subset R_3 \subset \cdots \subset CS$.
- $R_2 \subset CF$.
- $CF \not\subseteq R_n$, $n \geq 1$.
- For all $n \geq 1$, $R_n$ is closed under union, finite substitution, homomorphism, intersection with a regular language, and right quotient with a regular language.
- For all $n \geq 2$, $R_n$ is not closed under intersection and complement.

These statements and Theorem 9 imply statements 1, 2, 4, 5, 6 of Corollary 10. Moreover, observe that $\{a^n b^n c^{2n} : n \geq 0\} \in W_2 - CF$, which proves 3. □

**Theorem 11.** *For all $n \geq 1$, $W_n$ is not closed under inverse homomorphism.*

*Proof.* For $n = 1$, let $L = \{a^k b^k : k \geq 1\}$, and let the homomorphism $h : \{a, b, c\}^* \to \{a, b\}^*$ be defined as $h(a) = a$, $h(b) = b$, and $h(c) = \varepsilon$. Then, $L \in W_1$, but

$$L' = h^{-1}(L) \cap c^* a^* b^* = \{c^* a^k b^k : k \geq 1\} \notin W_1.$$

Assume that $L'$ is in $W_1$. Then, by Theorem 9, there is a 2-PRLG $G = (N_1, N_2, T, S, P)$ such that $\mathcal{L}(G) = L'$. Let $k > |P| \cdot \max\{|w| : X \to wY \in P\}$. Consider a derivation of $c^k a^k b^k \in L'$. The second component can generate only

finitely many $a$s; otherwise, it derives $\{a^k b^n : k < n\}$, which is not regular. Analogously, the first component generates only finitely many $b$s. Therefore, the first component generates any number of $a$s, and the second component generates any number of $b$s. Moreover, there is a derivation of the form $X \Rightarrow^m X$, for some $X \in N_2$, and $m \geq 1$, used in the derivation in the second component. In the first component, there is a derivation $A \Rightarrow^l a^s A$, for some $A \in N_1$, and $s, l \geq 1$. Then, we can modify the derivation of $c^k a^k b^k$ so that in the first component, we repeat the cycle $A \Rightarrow^l a^s A$ $(m+1)$-times, and in the second component, we repeat the cycle $X \Rightarrow^m X$ $(l+1)$-times. The derivations of both components have the same length—the added cycles are of length $ml$, and the rest is of the same length as in the derivation of $c^k a^k b^k$. Therefore, we have derived $c^k a^r b^k$, where $r > k$, which is not in $L'$—a contradiction.

For $n > 1$, the proof is analogous and left to the reader. □

**Corollary 12.** *For all $n \geq 1$, $W_n$ is not closed under concatenation. Therefore, it is not closed under Kleene closure either.*

*Proof.* For $n = 1$, let $L_1 = \{c\}^*$ and $L_2 = \{a^k b^k : k \geq 1\}$. Then, $L_1 L_2 = \{c^* a^k b^k : k \geq 1\}$. Analogously, prove this corollary for $n > 1$. □

## 4.2 $n$-Turn All-Move Self-Regulating Finite Automata

This section discusses $n$-turn all-move self-regulating finite automata. It proves that the family of languages accepted by $n$-all-SFAs coincides with the family of languages generated by $n$-RLSMGs.

**Lemma 13.** *For every $n$-RLSMG, $G = (N_1, \ldots, N_n, T, S, P)$, there is an equivalent $n$-RLSMG, $G'$, that satisfies:*

1. *if $[S \to X_1 \ldots X_n]$, then $X_i$ does not occur on the right-hand side of any rule, $1 \leq i \leq n$;*
2. *if $[S \to \alpha]$, $[S \to \beta] \in P$ and $\alpha \neq \beta$, then $alph(\alpha) \cap alph(\beta) = \emptyset$;*
3. *for any two matrices $m_1, m_2 \in P$, if $m_1[i] = m_2[i]$, for some $1 \leq i \leq n$, then $m_1 = m_2$.*

*Proof.* The first two conditions can be proved analogously to Lemma 6. Suppose that there are matrices $m$ and $m'$ such that $m[i] = m'[i]$, for some $1 \leq i \leq n$. Let $m = [X_1 \to x_1, \ldots, X_n \to x_n]$, $m' = [Y_1 \to y_1, \ldots, Y_n \to y_n]$. Replace these matrices with matrices $m_1 = [X_1 \to X_1', \ldots, X_n \to X_n']$, $m_2 = [X_1' \to x_1, \ldots, X_n' \to x_n]$, and $m_1' = [Y_1 \to Y_1'', \ldots, Y_n \to Y_n'']$, $m_2' = [Y_1'' \to y_1, \ldots, Y_n'' \to y_n]$, where $X_i'$, $Y_i''$ are new nonterminals for all $i$. These new matrices satisfy condition 3. Repeat this replacement until the resulting grammar satisfies the properties of $G'$ given in this lemma. □

**Lemma 14.** *Let $G$ be an $n$-RLSMG. There is an $(n-1)$-all-SFA, $M$, such that $\mathcal{L}(G) = \mathcal{L}(M)$.*

*Proof.* Without loss of generality, we assume that $G = (N_1, \ldots, N_n, T, S, P)$ is in the form described in Lemma 13. We construct $(n-1)$-all-SFA $M = (Q, T, \delta, q_0, q_t, F, R)$, where $Q = \{q_0, \ldots, q_n\} \cup N$, $N = N_1 \cup \cdots \cup N_n$, $\{q_0, q_1, \ldots, q_n\} \cap N = \emptyset$, $F = \{q_n\}$, $\delta = \{q_i \to X_{i+1} : [S \to X_1 \ldots X_n] \in P, 0 \leq i < n\} \cup \{X_i w_i \to Y_i : [X_1 \to w_1 Y_1, \ldots, X_n \to w_n Y_n] \in P, 1 \leq i \leq n\} \cup \{X_i w_i \to q_i : [X_1 \to w_1, \ldots, X_n \to w_n] \in P, w_i \in T^*, 1 \leq i \leq n\}$, $q_t = q_1$, $\Psi = \delta$ with the identity map, and $R = \{(q_i \to X_{i+1}, q_{i+1} \to X_{i+2}) : [S \to X_1 \ldots X_n] \in P, 0 \leq i \leq n-2\} \cup \{(X_i w_i \to Y_i, X_{i+1} w_{i+1} \to Y_{i+1}) : [X_1 \to w_1 Y_1, \ldots, X_n \to w_n Y_n] \in P, 1 \leq i < n\} \cup \{(X_i w_i \to q_i, X_{i+1} w_{i+1} \to q_{i+1}) : [X_1 \to w_1, \ldots, X_n \to w_n] \in P, w_i \in T^*, 1 \leq i < n\}$.

We next prove $\mathcal{L}(G) = \mathcal{L}(M)$. The proof of $\mathcal{L}(G) \subseteq \mathcal{L}(M)$ is very similar to the proof of the same inclusion of Lemma 7, so it is left to the reader.

To prove $\mathcal{L}(M) \subseteq \mathcal{L}(G)$, consider $w \in \mathcal{L}(M)$ and an acceptance of $w$ in $M$. As in Lemma 7, the derivation looks like the one depicted on the right-hand side of Fig. 3. Next, we generate $w$ in $G$ as follows. By Lemma 13, there is matrix $[S \to X_1^1 X_1^2 \ldots X_1^n]$ in $P$. Moreover, if $X_j^i x_j^i \to X_{j+1}^i$, $1 \leq i \leq n$, then $(X_j^i \to x_j^i X_{j+1}^i, X_j^{i+1} \to x_j^{i+1} X_{j+1}^{i+1}) \in R$, for $1 \leq i < n$, $1 \leq j < k$. We apply $[X_j^1 \to x_j^1 X_{j+1}^1, \ldots, X_j^n \to x_j^n X_{j+1}^n]$ from $P$. If $X_k^i x_k^i \to q_i$, $1 \leq i \leq n$, then $(X_k^i \to x_k^i, X_k^{i+1} \to x_k^{i+1}) \in R$, for $1 \leq i < n$, and we apply $[X_k^1 \to x_k^1, \ldots, X_k^n \to x_k^n] \in P$. Thus, $w \in \mathcal{L}(G)$.

Hence, Lemma 14 holds. □

**Lemma 15.** *Let $M$ be an $n$-all-SFA. There is an $(n+1)$-RLSMG, $G$, such that $\mathcal{L}(G) = \mathcal{L}(M)$.*

*Proof.* Let $M = (Q, \Sigma, \delta, q_0, q_t, F, R)$. Consider $G = (N_0, \ldots, N_n, \Sigma, S, P)$, where $N_i = (Q\Sigma^l \times Q \times \{i\} \times Q) \cup (Q \times \{i\} \times Q)$, $l = \max\{|w| : qw \to p \in \delta\}$, $0 \leq i \leq n$, and

$$
\begin{aligned}
P = \; & \{[S \to [q_0 x_0, q^0, 0, q_t][q_t x_1, q^1, 1, q_{i_1}] \ldots [q_{i_{n-1}} x_n, q^n, n, q_{i_n}]] : \\
& r_0.q_0 x_0 \to q^0, r_1.q_t x_1 \to q^1, \ldots, r_n.q_{i_{n-1}} x_n \to q^n \in \delta, \\
& (r_0, r_1), \ldots, (r_{n-1}, r_n) \in R, q_{i_n} \in F\} \cup \\
& \{[[p_0 x_0, q_0, 0, r_0] \to x_0[q_0, 0, r_0], \ldots, [p_n x_n, q_n, n, r_n] \to x_n[q_n, n, r_n]]\} \cup \\
& \{[[q_0, 0, q_0] \to \varepsilon, \ldots, [q_n, n, q_n] \to \varepsilon] : q_i \in Q, 0 \leq i \leq n\} \cup \\
& \{[[q_0, 0, p_0] \to w_0[q_0', 0, p_0], \ldots, [q_n, n, p_n] \to w_n[q_n', n, p_n]] : r_j.q_j w_j \to q_j' \in \\
& \delta, 0 \leq j \leq n, (r_i, r_{i+1}) \in R, 0 \leq i < n\}.
\end{aligned}
$$

Next, we prove $\mathcal{L}(G) = \mathcal{L}(M)$. To prove $\mathcal{L}(G) \subseteq \mathcal{L}(M)$, consider a derivation of $w$ in $G$. Then, the derivation is of the form (1) and there are rules $r_0.q_0 x_0^0 \to q_1^0, r_1.q_t x_0^1 \to q_1^1, \ldots, r_n.q_{i_{n-1}} x_0^n \to q_1^n$ in $\delta$ such that $(r_0, r_1), \ldots, (r_{n-1}, r_n) \in R$. Moreover, $(r_j^l, r_j^{l+1}) \in R$, where $r_j^l.q_j^l x_j^l \to q_{j+1}^l \in \delta$, and $(r_k^l, r_k^{l+1}) \in R$, where $r_k^l.q_k^l x_k^l \to q_{i_l} \in \delta$, $0 \leq l < n$, $1 \leq j < k$, $q_{i_0}$ denotes $q_t$, and $q_{i_n} \in F$. Thus, $M$ accepts $w$ with the list of rules $\mu$ of the form (2).

To prove $\mathcal{L}(M) \subseteq \mathcal{L}(G)$, let $\mu$ be a list of rules used in an acceptance of

$$w = x_0^0 x_1^0 \ldots x_k^0 x_0^1 x_1^1 \ldots x_k^1 \ldots x_0^n x_1^n \ldots x_k^n$$

in $M$ of the form (2). Then, the derivation is of the form (1) because

$$[[q_j^0, 0, q_t] \rightarrow x_j^0[q_{j+1}^0, 0, q_t], \ldots, [q_j^n, n, q_{i_n}] \rightarrow x_j^n[q_{j+1}^n, n, q_{i_n}]] \in P,$$

for all $q_j^i \in Q$, $1 \leq i \leq n$, $1 \leq j < k$, and $[[q_t, 0, q_t] \rightarrow \varepsilon, \ldots, [q_{i_n}, n, q_{i_n}] \rightarrow \varepsilon] \in P$.
Hence, Lemma 15 holds. □

The second main result of this paper follows next.

**Theorem 16.** *For all $n \geq 0$, $S_n = R_{[n+1]}$.*

*Proof.* This proof follows from Lemma 14 and 15. □

**Corollary 17.** *The following statements hold:*

1. $REG = S_0 \subset S_1 \subset S_2 \subset \cdots \subset CS$.
2. $S_1 \not\subseteq CF$.
3. $CF \not\subseteq S_n$, *for every* $n \geq 0$.
4. *For all $n \geq 0$, $S_n$ is closed under union, concatenation, finite substitution, homomorphism, intersection with a regular language, and right quotient with a regular language.*
5. *For all $n \geq 1$, $S_n$ is not closed under intersection, complement, and Kleene closure.*

*Proof.* Recall the following statements proved in [28]:

- $REG = R_{[1]} \subset R_{[2]} \subset R_{[3]} \subset \cdots \subset CS$.
- For all $n \geq 1$, $R_{[n]}$ is closed under union, finite substitution, homomorphism, intersection with a regular language, and right quotient with a regular language.
- For all $n \geq 2$, $R_{[n]}$ is not closed under intersection and complement.

Furthermore, recall these statements proved in [23] and [24]:

- For all $n \geq 1$, $R_{[n]}$ is closed under concatenation.
- For all $n \geq 2$, $R_{[n]}$ is not closed under Kleene closure.

These statements and Theorem 16 imply statements 1, 4, and 5 of Corollary 17. Moreover, observe that $\{ww : w \in \{a, b\}^*\} \in S_1 - CF$ (see *Example 5*), which proves 2. Finally, let $L = \{wcw^R : w \in \{a, b\}^*\}$. In [4, Theorem 1.5.2], there is a proof that $L \notin R_{[n]}$, for any $n \geq 1$. Thus, 3 follows from Theorem 16. □

Theorem 18, given next, follows from Theorem 16 and from Corollary 3.3.3 in [24]. However, Corollary 3.3.3 in [24] is not proved effectively. We next prove Theorem 18 effectively.

**Theorem 18.** *$S_n$ is closed under inverse homomorphism, for all $n \geq 0$.*

*Proof.* For $n = 1$, let $M = (Q, \Sigma, \delta, q_0, q_t, F, R)$ be a 1-all-SFA, and let $h : \Delta^* \to \Sigma^*$ be a homomorphism. Next, we construct 1-all-SFA $M' = (Q', \Delta, \delta', q_0', q_t', \{q_f'\}, R')$ accepting $h^{-1}(\mathcal{L}(M))$ as follows. Denote $k = \max\{|w| : qw \to p \in \delta\} + \max\{|h(a)| : a \in \Delta\}$. Let $Q' = q_0' \cup \{[x, q, y] : x, y \in \Sigma^*, |x|, |y| \leq k, q \in Q\}$. Initially, set $\delta'$ and $R'$ to $\emptyset$. Then, extend $\delta'$ and $R'$ by performing 1 through 5:

1. For $y \in \Sigma^*$, $|y| \leq k$, add
   $(q_0' \to [\varepsilon, q_0, y], q_t' \to [y, q_t, \varepsilon])$ to $R'$;
2. For $A \in Q'$, $q \neq q_t$, add
   $([x, q, y]a \to [xh(a), q, y], A \to A)$ to $R'$;
3. For $A \in Q'$, add
   $(A \to A, [x, q, \varepsilon]a \to [xh(a), q, \varepsilon])$ to $R'$;
4. For $(qx \to p, q'x' \to p') \in R$, $q \neq q_t$, add
   $([xw, q, y] \to [w, p, y], [x'w', q', \varepsilon] \to [w', p', \varepsilon])$ to $R'$;
5. For $q_f \in F$, add
   $([y, q_t, y] \to q_t', [\varepsilon, q_f, \varepsilon] \to q_f')$ to $R'$.

In essence, $M'$ simulates $M$ in the following way. In a state of the form $[x, q, y]$, the three components have the following meaning:

- $x = h(a_1 \ldots a_n)$, where $a_1 \ldots a_n$ is the input string that $M'$ has already read;
- $q$ is the current state of $M$;
- $y$ is the suffix remaining as the first component of the state that $M'$ enters during a turn; $y$ is thus obtained when $M'$ reads the last symbol right before the turn occurs in $M$; $M$ reads $y$ after the turn.

More precisely, $h(w) = w_1 y w_2$, where $w$ is an input string, $w_1$ is accepted by $M$ before making the turn, i.e. from $q_0$ to $q_t$, and $yw_2$ is accepted by $M$ after making the turn, i.e. from $q_t$ to $q_f \in F$. A rigorous version of this proof is left to the reader.

For $n > 1$, the proof is analogous and left to the reader. □

## 4.3 Language Families Accepted by $n$-first-SFAs and $n$-all-SFAs

In this section, we compare the family of languages accepted by $n$-first-SFAs with the family of languages accepted by $n$-all-SFAs.

**Theorem 19.** *For all $n \geq 1$, $W_n \subset S_n$.*

*Proof.* In [21] and [28], it is proved that for all $n > 1$, $R_n \subset R_{\lfloor n \rfloor}$. The proof of Theorem 19 thus follows from Theorem 9 and 16. □

**Theorem 20.** $W_n \not\subseteq S_{n-1}$, $n \geq 1$.

*Proof.* It is easy to see that $L = \{a_1^k a_2^k \dots a_{n+1}^k : k \geq 1\} \in W_n = R_{n+1}$. However, $L \notin S_{n-1} = R_{[n]}$ (see Lemma 1.5.6 in [4]). □

**Lemma 21.** *For each regular language, $L$, language $\{w^n : w \in L\} \in S_{n-1}$.*

*Proof.* Let $L = \mathcal{L}(M)$, where $M$ is a finite automaton. Make $n$ copies of $M$. Rename their states so all the sets of states are pairwise disjoint. In this way, also rename the states in the rules of each of these $n$ automata; however, keep the labels of the rules unchanged. For each rule label $r$, include $(r, r)$ into $R$. As a result, we obtain an $n$-turn all-move self-regulating finite automaton that accepts $\{w^n : w \in L\}$. A rigorous version of this proof is left to the reader. □

**Theorem 22.** *$S_n - W \neq \emptyset$, for all $n \geq 1$, where $W = \bigcup_{m=1}^{\infty} W_m$.*

*Proof.* By induction on $n \geq 1$, we prove that language $L = \{(cw)^{n+1} : w \in \{a, b\}^*\} \notin W$. From Lemma 21, $L \in S_n$.

**Basis:** For $n = 1$, let $G$ be an $m$-PRLG generating $L$, for some positive integer $m$. Consider a sufficiently large string $cw_1cw_2 \in L$ such that $w_1 = w_2 = a^{n_1}b^{n_2}$, $n_2 > n_1 > 1$. Then, there is a derivation of the form

$$\begin{aligned} S &\Rightarrow^p \\ x_1A_1x_2A_2 \dots x_mA_m &\Rightarrow^k x_1y_1A_1x_2y_2A_2 \dots x_my_mA_m \end{aligned} \tag{3}$$

in $G$, where cycle (3) generates more than one $a$ in $w_1$. The derivation continues as

$$\begin{aligned} x_1y_1A_1 \dots x_my_mA_m &\Rightarrow^r \\ x_1y_1z_1B_1 \dots x_my_mz_mB_m &\Rightarrow^l x_1y_1z_1u_1B_1 \dots x_my_mz_mu_mB_m \\ (\text{cycle (4) generates no } a\text{s}) &\Rightarrow^s cw_1cw_2. \end{aligned} \tag{4}$$

Next, modify the left derivation, the derivation in components generating $cw_1$, so that the $a$-generating cycle (3) is repeated $(l+1)$-times. Similarly, modify the right derivation, the derivation in the other components, so that the no-$a$-generating cycle (4) is repeated $(k+1)$-times. Thus, the modified left derivation is of length $p + k(l+1) + r + l + s = p + k + r + l(k+1) + s$, which is the length of the modified right derivation. Moreover, the modified left derivation generates more $a$s in $w_1$ than the right derivation in $w_2$—a contradiction.

**Induction step:** Suppose that the theorem holds for all $n \leq k$, for some $k \geq 1$. Consider $n+1$ and let $\{(cw)^{n+1} : w \in \{a, b\}^*\} \in W_l$, for some $l \geq 1$. As $W_l$ is closed under the right quotient with a regular language, and language $\{cw : w \in \{a, b\}^*\}$ is regular, we obtain $\{(cw)^n : w \in \{a, b\}^*\} \in W_l \subseteq W$—a contradiction. □

Fig. 4 summarizes the language families discussed in this paper.

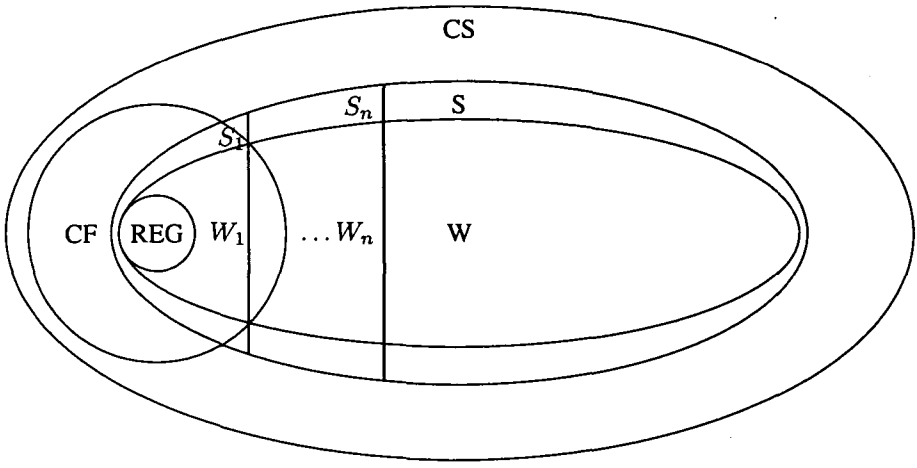


Figure 4: The hierarchy of languages.

# 5 Conclusion and Discussion

This paper has discussed self-regulating finite automata. As demonstrated next, we can analogically introduce and discuss self-regulating pushdown automata.

Recall that a *pushdown automaton* (see [15]), $M$, is a septuple $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, where $Q$, $\Sigma$, $q_0 \in Q$, $F$ are as in a finite automaton, $\Gamma$ is a finite pushdown alphabet, $\delta$ is a finite set of rules of the form $Zqw \rightarrow \gamma p$, $q, p \in Q$, $Z \in \Gamma$, $w \in \Sigma^*$, $\gamma \in \Gamma^*$, and $Z_0$ is an initial pushdown symbol. Again, let $\psi$ denote the bijection from $\delta$ to $\Psi$, and write $r.Zqw \rightarrow \gamma p$ instead of $\psi(Zqw \rightarrow \gamma p) = r$. A configuration of $M$ is any word from $\Gamma^* Q \Sigma^*$. For any configuration $xAqwy$, where $x \in \Gamma^*$, $y \in \Sigma^*$, $q \in Q$, and any $r.Aqw \rightarrow \gamma p \in \delta$, $M$ makes a move from $xAqwy$ to $x\gamma py$ according to $r$, written as $xAqwy \Rightarrow x\gamma py\,[r]$. As usual, we define closure $\Rightarrow^*$. If $w \in \Sigma^*$ and $Z_0 q_0 w \Rightarrow^* f\,[\mu]$, $f \in F$, then $w$ is accepted by $M$ and $Z_0 q_0 w \Rightarrow^* f\,[\mu]$ is an acceptance of $w$ in $M$. The language of $M$ is defined as $\mathcal{L}(M) = \{w \in \Sigma^* : Z_0 q_0 w \Rightarrow^* f\,[\mu] \text{ is an acceptance of } w\}$.

**Definition 23.** *A* self-regulating pushdown automaton, *SPDA, $M$, is a nonuple*

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_t, Z_0, F, R),$$

*where*

1. $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ *is a pushdown automaton,*
2. $q_t \in Q$ *is a* turn state, *and*
3. $R \subseteq \Psi \times \Psi$ *is a* finite relation, *where $\Psi$ is an alphabet of rule labels.*

**Definition 24.** *Let $n \geq 0$ and $M = (Q, \Sigma, \Gamma, \delta, q_0, q_t, Z_0, F, R)$ be a self-regulating pushdown automaton. $M$ is said to be an* $n$-turn first-move self-regulating pushdown automaton, *$n$-first-SPDA, if $M$ accepts $w$ in the following way. There is an acceptance $Z_0 q_0 w \Rightarrow^* f\,[\mu]$ such that*

$$\mu = r_1^0 \ldots r_k^0 r_1^1 \ldots r_k^1 \ldots r_1^n \ldots r_k^n,$$

*where $k \in \mathbb{N}$, $r_k^0$ is the first rule of the form $Zqx \rightarrow \gamma q_t$, for some $Z \in \Gamma$, $q \in Q$, $x \in \Sigma^*$, $\gamma \in \Gamma^*$, and*

$$(r_1^j, r_1^{j+1}) \in R$$

*for all $0 \leq j < n$.*

*The family of languages accepted by $n$-first-SPDAs is denoted by $\mathcal{L}(n\text{-first-SPDA})$.*

**Definition 25.** *Let $n \geq 0$ and $M = (Q, \Sigma, \Gamma, \delta, q_0, q_t, Z_0, F, R)$ be a self-regulating pushdown automaton. $M$ is said to be an* $n$-turn all-move self-regulating pushdown automaton, *$n$-all-SPDA, if $M$ accepts $w$ in the following way. There is an acceptance $Z_0 q_0 w \Rightarrow^* f\,[\mu]$ such that*

$$\mu = r_1^0 \ldots r_k^0 r_1^1 \ldots r_k^1 \ldots r_1^n \ldots r_k^n,$$

*where $k \in \mathbb{N}$, $r_k^0$ is the first rule of the form $Zqx \rightarrow \gamma q_t$, for some $Z \in \Gamma$, $q \in Q$, $x \in \Sigma^*$, $\gamma \in \Gamma^*$, and*

$$(r_i^j, r_i^{j+1}) \in R$$

*for all $1 \leq i \leq k$, $0 \leq j < n$.*

*The family of languages accepted by $n$-all-SPDAs is denoted by $\mathcal{L}(n\text{-all-SPDA})$.*

## 5.1 $n$-Turn All-Move Self-Regulating Pushdown Automata

It is easy to see that an $n$-turn all-move self-regulating pushdown automaton without any turn state is exactly a common pushdown automaton. Therefore, $\mathcal{L}(0\text{-all-SPDA}) = CF$. Moreover, if we consider 1-turn all-move self-regulating pushdown automata, their power is that of the Turing machines.

**Theorem 26.** $\mathcal{L}(1\text{-}all\text{-}SPDA) = RE$.

*Proof.* For any $L \in RE$, $L \subseteq \Delta^*$, there are context-free languages $\mathcal{L}(G)$ and $\mathcal{L}(H)$ and a homomorphism $h : \Sigma^* \rightarrow \Delta^*$ such that $L = h(\mathcal{L}(G) \cap \mathcal{L}(H))$ (see Theorem 1.12 in [14]). Suppose that $G = (N_G, \Sigma, P_G, S_G)$, $H = (N_H, \Sigma, P_H, S_H)$ are in the Greibach normal form, i.e. all rules are of the form $A \rightarrow a\alpha$, where $A$ is a nonterminal, $a$ is a terminal, and $\alpha$ is a (possibly empty) string of nonterminals. Let us construct 1-all-SPDA $M = (\{q_0, q, q_t, p, f\}, \Delta, \Sigma \cup N_G \cup N_H \cup \{Z\}, \delta, q_0, Z, \{f\}, R)$, $Z \notin \Sigma \cup N_G \cup N_H$, with $R$ made as follows:

1. add $(Zq_0 \rightarrow ZS_G q, Zq_t \rightarrow ZS_H p)$ to $R$

2. add $(Aq \rightarrow B_n \dots B_1 aq, Cp \rightarrow D_m \dots D_1 ap)$ to $R$ if
   $A \rightarrow aB_1 \dots B_n \in P_G$ and
   $C \rightarrow aD_1 \dots D_m \in P_H$
3. add $(aqh(a) \rightarrow q, ap \rightarrow p)$ to $R$
4. add $(Zq \rightarrow Zq_t, Zp \rightarrow f)$ to $R$

Moreover, $\delta$ contains only the rules from the definition of $R$.

Now, we prove $w \in h(\mathcal{L}(G) \cap \mathcal{L}(H))$ if and only if $w \in \mathcal{L}(M)$.

*Only if Part:* Let $w \in h(\mathcal{L}(G) \cap \mathcal{L}(H))$. There are $a_1, a_2, \dots, a_n \in \Sigma$ such that $a_1 a_2 \dots a_n \in \mathcal{L}(G) \cap \mathcal{L}(H)$ and $w = h(a_1 a_2 \dots a_n)$, for some $n \geq 0$. There are leftmost derivations $S_G \Rightarrow^n a_1 a_2 \dots a_n$ and $S_H \Rightarrow^n a_1 a_2 \dots a_n$ of length $n$ in $G$ and $H$, respectively, because in every derivation step exactly one terminal element is derived. Thus, $M$ accepts $h(a_1) \dots h(a_n)$ as

$$Zq_0 h(a_1) \dots h(a_n) \Rightarrow ZS_G q h(a_1) \dots h(a_n), \dots, Za_n q h(a_n) \Rightarrow Zq,\ Zq \Rightarrow Zq_t,$$

$$Zq_t \Rightarrow ZS_H p, \dots, Za_n p \Rightarrow Zp,\ Zp \Rightarrow f.$$

In state $q$, by using its pushdown, $M$ simulates $G$'s derivation of $a_1 \dots a_n$ but reads $h(a_1) \dots h(a_n)$ as the input. In $p$, $M$ simulates $H$'s derivation of $a_1 a_2 \dots a_n$ but reads no input. As $a_1 a_2 \dots a_n$ can be derived in both $G$ and $H$ by making the same number of steps, the automaton can successfully complete the acceptance of $w$.

*If Part:* Notice that in one step, $M$ can read only $h(a) \in \Delta^*$, for some $a \in \Sigma$. Let $w \in \mathcal{L}(M)$, then $w = h(a_1) \dots h(a_n)$, for some $a_1, \dots, a_n \in \Sigma$. Consider $M$'s acceptance of $w$

$$Zq_0 h(a_1) \dots h(a_n) \Rightarrow ZS_G q h(a_1) \dots h(a_n), \dots, Za_n q h(a_n) \Rightarrow Zq,\ Zq \Rightarrow Zq_t,$$

$$Zq_t \Rightarrow ZS_H p, \dots, Za_n p \Rightarrow Zp,\ Zp \Rightarrow f.$$

As stated above, in $q$, $M$ simulates $G$'s derivation of $a_1 a_2 \dots a_n$, and then in $p$, $M$ simulates $H$'s derivation of $a_1 a_2 \dots a_n$. It successfully completes the acceptance of $w$ only if $a_1 a_2 \dots a_n$ can be derived in both $G$ and $H$. Hence, the if part holds, too. □

## 5.2 Open Problems

Although the fundamental results about self-regulating automata have been achieved in this paper, there still remain several open problems concerning them. Perhaps most importantly, these open problem areas include 1 through 3 given next:

1. What is the language family accepted by $n$-turn first-move self-regulating pushdown automata, when $n \geq 1$ (see Definition 24)?
2. By analogy with the standard deterministic finite and pushdown automata (see page 145 and page 437 in [15]), introduce the deterministic versions of self-regulating automata. What is their power?

3. Discuss the closure properties of other language operations, such as the reversal.

## Acknowledgements

The authors would like to thank both anonymous referees for their suggestions. This work was supported by the GAČR 201/07/0005, 102/05/H050, and FR762/2007/G1 grants.

# References

[1] A. V. Aho and J. D. Ullman. *The Theory of Parsing, Translation and Compiling, Volume I: Parsing.* Prentice Hall, Englewood Cliffs, New Jersey, 1972.

[2] J. M. Autebert, J. Berstel, and L. Boasson. Context-free languages and pushdown automata. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, Word Language Grammar, pages 111–174. Springer-Verlag, Berlin, 1997.

[3] B. Courcelle. On jump deterministic pushdown automata. *Math. Systems Theory*, 11:87–109, 1977.

[4] J. Dassow and Gh. Păun. *Regulated Rewriting in Formal Language Theory.* Springer-Verlag, Berlin, 1989.

[5] P. C. Fischer and A. L. Rosenberg. Multitape one-way nonwriting automata. *J. Comput. System Sci.*, 2:38–101, 1968.

[6] S. Ginsburg, S. A. Greibach, and M. A. Harrison. One-way stack automata. *J. ACM*, 14:389–418, 1967.

[7] S. Ginsburg and E. Spanier. Finite-turn pushdown automata. *SIAM J. Control*, 4:429–453, 1968.

[8] S. A. Greibach. Checking automata and one-way stack languages. *J. Comput. System Sci.*, 3:196–217, 1969.

[9] M. A. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley, Reading, Massachusetts, 1978.

[10] O. H. Ibarra. Simple matrix languages. *Inform. and Control*, 17:359–394, 1970.

[11] M. Ito. *Algebraic Theory of Automata and Languages.* World Scientific, Singapore, 2004.

[12] H. R. Lewis and C. H. Papadimitriou. *Elements of the Theory of Computation.* Prentice-Hall, Englewood Cliffs, 1981.

[13] J. C. Martin. *Introduction to Languages and the Theory of Computation.* McGraw-Hill, New York, 1991.

[14] A. Mateescu and A. Salomaa. Aspects of classical language theory. In G. Rozenberg and A. Salomaa, editors, *Handbook of formal languages*, volume 1, Word Language Grammar, pages 175–251. Springer-Verlag, Berlin, 1997.

[15] A. Meduna. *Automata and Languages: Theory and Applications.* Springer, London, 2000.

[16] A. Meduna. Simultaneously one-turn two-pushdown automata. *Int. J. Comp. Math.*, 80:679–687, 2003.

[17] A. Meduna and D. Kolář. Regulated pushdown automata. *Acta Cybernet.*, 14:653–664, 2000.

[18] A. Meduna and D. Kolář. One-turn regulated pushdown automata and their reduction. *Fund. Inform.*, 51:399–405, 2002.

[19] G. E. Revesz. *Introduction to Formal Languages.* McGraw-Hill, New York, 1983.

[20] R. D. Rosebrugh and D. Wood. A characterization theorem for $n$-parallel right linear languages. *J. Comput. System Sci.*, 7:579–582, 1973.

[21] R. D. Rosebrugh and D. Wood. Restricted parallelism and right linear grammars. *Util. Math.*, 7:151–186, 1975.

[22] J. Sakarovitch. Pushdown automata with terminating languages. *Languages and Automata Symposium, RIMS 421, Kyoto University*, pages 15–29, 1981.

[23] R. Siromoney. *Studies in the Mathematical Theory of Grammars and its Applications.* PhD thesis, University of Madras, Madras, India, 1969.

[24] R. Siromoney. Finite-turn checking automata. *J. Comput. System Sci.*, 5:549–559, 1971.

[25] T. A. Sudkamp. *Languages and Machines.* Addison-Wesley, Reading, Massachusetts, 1988.

[26] L. Valiant. The equivalence problem for deterministic finite turn pushdown automata. *Inform. and Control*, 81:265–279, 1989.

[27] D. Wood. Properties of $n$-parallel finite state languages. Technical report, McMaster University, 1973.

[28] D. Wood. $m$-parallel $n$-right linear simple matrix languages. *Util. Math.*, 8:3–28, 1975.

*Received June, 2006*

Acta Cybernetica **18** (2007) 155–165.

# Automata with Finite Congruence Lattices*

István Babcsányi†

*To the memory of Balázs Imreh*

**Abstract**

In this paper we prove that if the congruence lattice of an automaton $\mathbf{A}$ is finite then the endomorphism semigroup $E(\mathbf{A})$ of $\mathbf{A}$ is finite. Moreover, if $\mathbf{A}$ is commutative then $\mathbf{A}$ is A-finite. We prove that if $3 \leq |A|$ then a commutative automaton $\mathbf{A}$ is simple if and only if it is a cyclic permutation automaton of prime order. We also give some results concerning cyclic, strongly connected and strongly trap-connected automata.

## 1 Preliminaries

In this paper, by an *automaton* $\mathbf{A} = (A, X, \delta)$ we mean always an automaton without outputs, where $A \neq \emptyset$ is the *state set* and $X \neq \emptyset$ is the *input set.* Denote $|A|$ the cardinality of the set $A$. The automaton $\mathbf{A}$ is called *A-finite* if $|A| < \infty$. If $|A| = n$ then we say that $n$ is the *order of* $\mathbf{A}$ and if $n$ is a prime then $\mathbf{A}$ is an *automaton of prime order.* The *input monoid* [*semigroup*] $X^*$ [$X^+$] of $\mathbf{A}$ is the free monoid [semigroup] over $X$. The *transition function* $\delta : A \times X \to A$ can be extended in the usual way. If $e \in X^*$ is the empty word then let $\delta(a, e) = a$ for every $a \in A$; if $a \in A$, $p \in X^*$ and $x \in X$ then let $\delta(a, px) = \delta(\delta(a, p), x)$. Sometimes, we shall use the notation $ap$ instead of $\delta(a, p)$.

As known, every automaton can be considered as a unary algebra. Thus the notions such as subautomaton, congruence, homomomorphism, isomorphism etc. can be introduced in the following natural way.

An equivalence relation $\rho$ of state set $A$ of the automaton $\mathbf{A}$ is called a *congruence* on $\mathbf{A}$ if

$$(a, b) \in \rho \Longrightarrow (ax, bx) \in \rho,$$

for all $a, b \in A$ and $x \in X$. The $\rho$-class of $\mathbf{A}$ containing the state $a$ is denoted by $\rho[a]$. Denote $C(\mathbf{A})$ the congruence lattice of $\mathbf{A}$. Let $\iota_A$ [$\omega_A$] be the equality [universal] relation on $A$. The automaton $\mathbf{A}$ is called *simple* if $C(\mathbf{A}) = \{\iota_A, \omega_A\}$. It is evident that if $|A| \leq 2$ then $\mathbf{A}$ is simple.

*Research supported by the Hungarian NFSR grant No 67639.

†Department of Algebra, Mathematical Institute, Budapest University of Technology and Economics, 1111 Budapest, Egry József u. 1, Hungary. E-mail: babcs@math.bme.hu

The automaton $\mathbf{A}' = (A', X, \delta')$ is a *subautomaton* of the automaton $\mathbf{A} = (A, X, \delta)$ if $A' \subseteq A$ and $\delta'$ is the restriction of $\delta$ to $A' \times X$. The congruence

$$\rho_{A'} = \{(a, b) \in A^2;\ a = b \text{ or } a, b \in A'\}$$

is called *the Rees congruence of* $\mathbf{A}$ *induced by* $\mathbf{A}'$ ([2]). The set $R(\mathbf{A})$ of Rees congruences of $\mathbf{A}$ is a sublattice of $C(\mathbf{A})$. It is called *the Rees congruence lattice* of $\mathbf{A}$.

Let $\mathbf{A} = (A, X, \delta)$ and $\mathbf{B} = (B, X, \delta')$ be arbitrary automata. We say that a mapping $\varphi : A \to B$ is a *homomorphism* of $\mathbf{A}$ into $\mathbf{B}$ if

$$\varphi(ax) = \varphi(a)x,$$

for all $a \in A$ and $x \in X$. The *kernel* of $\varphi$ is the congruence $\operatorname{Ker}\varphi$ defined by $(a, b) \in \operatorname{Ker}\varphi$ if and only if $\varphi(a) = \varphi(b)$ $(a, b \in A)$. If $A = B$ then $\varphi$ is an *endomorphism* of $\mathbf{A}$. Furthermore, if $\varphi$ is bijective then it is an *automorphism* of $\mathbf{A}$. The set $E(\mathbf{A})$ $[G(\mathbf{A})]$ of all endomorphisms [automorphisms] of $\mathbf{A}$ is a monoid [group] under the usual multiplication of mappings. $E(A)$ $[G(A)]$ is called the *endomorphism semigroup* [*automorphism group*] of $\mathbf{A}$.

For notations and notions not defined here we refer to the books P.M. Cohn [5], F. Gécseg [7], F. Gécseg, F. and I. Peák [8], K.H. Kim and F.W. Roush [10] and G. Lallement [11].

## 2 Automata with finite congruence lattices

Let $B$ be a nonempty subset of the state set $A$ of an automaton $\mathbf{A} = (A, X, \delta)$. Denote $[\mathbf{B}] = ([B], X, \delta')$ the subautomaton of $\mathbf{A}$ generated by $B$, that is, $[B] = \{bp; b \in B, p \in X^*\}$. Specially, denote $[\mathbf{a}] = ([a], X, \delta')$ the subautomaton generated by $a \in A$. If $A = [B]$ then $B$ is called a *generating set* of $\mathbf{A}$. If there exists a finite generating set of $\mathbf{A}$ then we say that $\mathbf{A}$ is *finitely generated.* Specially, if there exists a generating set containing only one element $a$ then $\mathbf{A}$ is called a *cyclic automaton* and we say that $a$ is a *generating element* of $\mathbf{A}$.

**Lemma 1.** *If the congruence lattice of an automaton* $\mathbf{A}$ *is finite then* $\mathbf{A}$ *has finitely many subautomata and the congruence lattices of its subautomata are also finite.*

*Proof.* Assume that the congruence lattice $C(\mathbf{A})$ of the automaton $\mathbf{A} = (A, X, \delta)$ is finite. Thus the Rees congruence lattice $R(\mathbf{A})$ is finite. From this it follows that $\mathbf{A}$ has finitely many subautomata.

If $\mathbf{A}' = (A', X, \delta')$ is a subautomaton of $\mathbf{A}$ and $\rho \in C(\mathbf{A}')$ then $\rho \cup \iota_A \in C(\mathbf{A})$. Furthermore, if $\rho, \rho' \in C(A')$ and $\rho \neq \rho'$ then $\rho \cup \iota_A \neq \rho' \cup \iota_A$. Thus $C(\mathbf{A}')$ is also finite. □

**Corollary 2.** *If the congruence lattice of an automaton is finite then the automaton is finitely generated.*

*Proof.* If the congruence lattice of an automaton is finite then by Lemma 1, the number of its subautomata and thus the number of its cyclic subautomata is finite. Therefore, the automaton is finitely generated. □

S. Radeleczki has prowed in [15] that if the congruence lattice of a unary algebra is finite then its automorphism group is finite, too. The following theorem is a generalization of this result.

**Theorem 3.** *If the congruence lattice $C(\mathbf{A})$ of an automaton $\mathbf{A} = (A, X, \delta)$ is finite then the endomorphism semigroup $E(\mathbf{A})$ is finite.*

*Proof.* First, we show that the automorphism group $G(\mathbf{A})$ is finite. Assume that the order of $\alpha \in G(\mathbf{A})$ is infinite. For every positive integer $m$, we define the binary relation $\rho_{\alpha^m}$ on $A$, as follows. For $a, b \in A$, $(a, b) \in \rho_{\alpha^m}$ if and only if there is an element $c$ of $A$ and there are integers $i, k, l$ such that $0 \leq i \leq m-1$ and

$$a = \alpha^{km+i}(c), \quad b = \alpha^{lm+i}(c).$$

It can be easily verified that $\rho_{\alpha^m}$ is a congruence of $\mathbf{A}$. Furthermore, if $m \neq n$ then $\rho_{\alpha^m} \neq \rho_{\alpha^n}$ in a contradiction with our assumption that the congruence lattice $C(\mathbf{A})$ is finite. Thus the order of every $\alpha \in G(\mathbf{A})$ is finite.

Let $r$ be the order of $\alpha \in G(\mathbf{A})$. Take the binary relation $\rho_\alpha$ on $A$ for which $(a, b) \in \rho_\alpha$ if and only if there are $c \in A$ and integers $0 \leq i, j \leq r-1$ such that

$$a = \alpha^i(c), \quad b = \alpha^j(c).$$

For every $\alpha \in G(\mathbf{A})$, the relation $\rho_\alpha$ is a congruence of $\mathbf{A}$. Assume that

$$\rho_\alpha = \rho_\beta, \quad \beta \in G(\mathbf{A}).$$

By Corollary 2, the automaton $\mathbf{A}$ is finitely generated. If $\{c_1, c_2, \ldots, c_k\}$ is a finite generating set of $\mathbf{A}$ then

$$\rho_\beta[c_1] = \rho_\alpha[c_1], \ \rho_\beta[c_2] = \rho_\alpha[c_2], \ldots, \rho_\beta[c_k] = \rho_\alpha[c_k],$$

that is,

$$\beta(c_1) = \alpha^{i_1}(c_1), \ \beta(c_2) = \alpha^{i_2}(c_2), \ldots, \beta(c_k) = \alpha^{i_k}(c_k)$$

$(0 \leq i_1, i_2, \ldots, i_k \leq r-1)$. This means that $\beta = \alpha^{i_j}$ on $[c_j]$ $(j = 1, 2, \ldots, k)$. From this it follows that the number of such $\beta$ is finite for arbitrary $\alpha \in G(\mathbf{A})$. Since $C(\mathbf{A})$ is finite, the number of different $\rho_\alpha$'s is finite. From these results it follows that $G(\mathbf{A})$ is finite.

Now we show that the endomorphism semigroup $E(\mathbf{A})$ is also finite. If $\alpha \in E(\mathbf{A})$ then $\mathbf{A}_\alpha = (\alpha(A), X, \delta')$ is a subautomaton of $\mathbf{A}$, where $\alpha(A) = \{\alpha(a); a \in A\}$. Let $\beta \in E(\mathbf{A})$ such that

$$\operatorname{Ker}\beta = \operatorname{Ker}\alpha \quad \text{and} \quad \beta(A) = \alpha(A).$$

Define the mapping $\varphi_{\alpha,\beta} : \alpha(A) \to \beta(A)$ such that

$$\varphi_{\alpha,\beta}(\alpha(a)) = \beta(a)$$

for every $a \in A$. This means that

$$\varphi_{\alpha,\beta}\alpha = \beta.$$

Since $\operatorname{Ker}\beta = \operatorname{Ker}\alpha$, $\varphi_{\alpha,\beta}$ is a bijective mapping. If $a \in A$ and $x \in X$ then

$$\varphi_{\alpha,\beta}(\alpha(a)x) = \varphi_{\alpha,\beta}\alpha(ax) = \beta(ax) = \beta(a)x = \varphi_{\alpha,\beta}(\alpha(a))x,$$

that is, $\varphi_{\alpha,\beta} \in G(\mathbf{A}_\alpha)$. By Lemma 1, $C(\mathbf{A}_\alpha)$ is finite and thus, by the first part of this proof, $G(\mathbf{A}_\alpha)$ is finite. Furthermore, if

$$\operatorname{Ker}\beta = \operatorname{Ker}\beta' = \operatorname{Ker}\alpha, \quad \beta(A) = \beta'(A) = \alpha(A)$$

and

$$\varphi_{\alpha,\beta} = \varphi_{\alpha,\beta'},$$

then $\beta = \beta'$. Thus, for arbitrary $\alpha \in E(\mathbf{A})$, the number of $\beta \in E(\mathbf{A})$ such that $\operatorname{Ker}\beta = \operatorname{Ker}\alpha$ and $\beta(A) = \alpha(A)$ is finite. Since the number of different $\operatorname{Ker}\alpha$'s and different $\beta(A)$'s $(\alpha, \beta \in E(\mathbf{A}))$ is finite, $E(\mathbf{A})$ is also finite. □

For every $a \in A$, consider the binary relation $\rho_{A,a}$ on $X^*$ defined as

$$(p,q) \in \rho_{A,a} \iff ap = aq \quad (p,q \in X^*).$$

It is clear that $\rho_{A,a}$ $(a \in A)$ is a right congruence on $X^*$. The relation $\rho_A = \cap_{a\in A}\rho_{A,a}$ is congruence on $X^*$. The *characteristic semigroup* $S(\mathbf{A})$ of the automaton $\mathbf{A}$ is the factor semigroup $X^*/\rho_A$.

R.H. Oehmke has shown in [13] the first part of the following lemma, that is, for arbitrary cyclic automaton $\mathbf{A} = (A, X, \delta)$, $|E(\mathbf{A})| \leq |A|$. We have shown in our paper [1] that $|A| \leq |S(\mathbf{A})|$.

**Lemma 4.** *For every cyclic automaton* $\mathbf{A} = (A, X, \delta)$,

$$|E(\mathbf{A})| \leq |A| \leq |S(\mathbf{A})|.$$

*Proof.* If $a_0$ is a generating element of $\mathbf{A}$ and $\alpha(a_0) = \beta(a_0)$ $(\alpha, \beta \in E(\mathbf{A}))$ then, for every $p \in X^*$,

$$\alpha(a_0p) = \alpha(a_0)p = \beta(a_0)p = \beta(a_0p),$$

that is, $\alpha = \beta$. Thus the mapping $\varphi : E(\mathbf{A}) \to A$ such that $\varphi(\alpha) = \alpha(a_0)$, for every $\alpha \in E(\mathbf{A})$, is an injective mapping of $E(\mathbf{A})$ into $A$. This means that $|E(\mathbf{A})| \leq |A|$.

If $a_0p \neq a_0q$ $(p,q \in X^*)$ then $\rho_A[p] \neq \rho_A[q]$. From this it follows that $|A| \leq |S(\mathbf{A})|$. □

**Lemma 5.** *If the relation* $\rho_{A,a_0}$ *is a congruence on* $X^*$, *for a generating element* $a_0$ *of a cyclic automaton* $\mathbf{A} = (A, X, \delta)$, *then* $E(\mathbf{A}) \cong S(\mathbf{A})$ *and* $|E(\mathbf{A})| = |A|$.

*Proof.* If the relation $\rho_{A,a_0}$ is a congruence on $X^*$ then $\rho_{A,a_0} = \rho_A$. Define the mapping $\alpha_p : A \to A$, for every $p \in X^*$, such that

$$\alpha_p(a_0q) = a_0pq \quad (q \in X^*).$$

It can easily be shown that $\alpha_p \in E(\mathbf{A})$. Furthermore, if $\alpha \in E(\mathbf{A})$ and $\alpha(a_0) = a_0r$ $(r \in X^*)$ then $\alpha = \alpha_r$. The mapping $\varphi : E(\mathbf{A}) \to S(\mathbf{A})$ such that

$$\varphi(\alpha_p) = \rho_A[p] \quad (p \in X^*)$$

is an isomorphism of $E(\mathbf{A})$ onto $S(\mathbf{A})$. By Lemma, $|E(\mathbf{A})| = |A|$. □

From Theorem 3, Lemma 4 and Lemma 5, we get the following corollary.

**Corollary 6.** *Let the congruence lattice $C(\mathbf{A})$ of the cyclic automaton $\mathbf{A} = (A, X, \delta)$ be finite. If the relation $\rho_{A,a_0}$ is a congruence on $X^*$, for a generating element $a_0$, then $\mathbf{A}$ is A-finite.*

The automaton $\mathbf{A}$ is *commutative* if $apq = aqp$ for every $a \in A$ and $p, q \in X^*$. It is immediate that every subautomaton of a commutative automaton is also commutative. I. Peák proved in [14] that $E(\mathbf{A}) \cong S(\mathbf{A})$ and $|E(\mathbf{A})| = |A|$ for arbitrary cyclic commutative automaton $\mathbf{A}$. (See also F. Gécseg and I. Peák [8], Z. Ésik and B. Imreh [6].) The statement of Lemma 5 is a generalization of this result. A.P. Grillet showed in [9] that if the congruence lattice of a commutative semigroup $S$ is finite then $S$ is finite. The following theorem generalizes this statement for commutative automata.

**Theorem 7.** *If the congruence lattice $C(\mathbf{A})$ of a commutative automaton $\mathbf{A} = (A, X, \delta)$ is finite then the automaton $\mathbf{A}$ is A-finite.*

*Proof.* By Corollary 2, $\mathbf{A}$ is finitely generated. Then, it is a union of commutative cyclic subautomata $\mathbf{A}_i = (A_i, X, \delta_i)$ $(i = 1, 2, \ldots, n)$. But, if $a_i \in A_i$ is a generating element of $\mathbf{A}_i$ then $\rho_{A_i,a_i}$ is a congruence on $X^*$, since $\mathbf{A}_i$ $(i = 1, 2, \ldots, n)$ is commutative. By Corollary 6, $\mathbf{A}_i$ $(i = 1, 2, \ldots, n)$ is A-finite and thus $\mathbf{A}$ is also finite. □

# 3 Simple automata

We discussed in our papers [3] and [4] the simple Mealy and Moore automata. In this paper we investigate the simplicity of the automata $\mathbf{A} = (A, X, \delta)$ without outputs. In this case $C(\mathbf{A}) = \{\iota_A, \omega_A\}$.

Let $H \neq \emptyset$ be a subset of the state set $A$ and let $Hp = \{ap; a \in H\}$ for every $p \in X^*$. Define the binary relation $\tau_H$ on $A$ as follows.

$$(a, b) \in \tau_H \quad \text{if and only if} \quad (ap \in H \iff bp \in H)$$

for every $p \in X^*$. $\tau_H$ is a congruence of $\mathbf{A}$ and $H$ is a union of certain $\tau_H$-congruence classes. The state $a \in A$ is called *disjunctive*, if $\tau_{\{a\}} = \iota_A$.

The set $H$ is called a *separator* of $\mathbf{A}$ if, for every $p \in X^*$,

$$Hp \subseteq H \quad \text{or} \quad Hp \cap H = \emptyset.$$

The one-element subsets of $A$ and itself $A$ are separators of $\mathbf{A}$. We say that these separators are the *trivial separators.*

**Lemma 8.** *The automaton $\mathbf{A} = (A, X, \delta)$ is simple if and only if every separator of $\mathbf{A}$ is trivial.*

*Proof.* Assume that all separators of $\mathbf{A}$ are trivial. If $\rho$ is a congruence of $\mathbf{A}$ then every $\rho$-class is a separator of $\mathbf{A}$. Therefore, $\rho = \iota_A$ or $\rho = \omega_A$, that is, $\mathbf{A}$ is a simple automaton.

Conversely, assume that $\mathbf{A}$ is simple. If $H$ is a separator of $\mathbf{A}$ then $\tau_H$ is a congruence of $\mathbf{A}$ such that $H$ is a $\tau_H$-class. But $\tau_H = \iota_A$ or $\tau_H = \omega_A$. Thus $|H| = 1$ or $H = A$ therefore $H$ is a trivial separator of $\mathbf{A}$. □

If every state of an automaton $\mathbf{A} = (A, X, \delta)$ is a generating element of $\mathbf{A}$ then we say that $\mathbf{A}$ is *strongly connected.* In other words, $\mathbf{A}$ is *strongly connected* if, for arbitrary states $a, b \in A$, there exists a $p \in X^+$ such that $ap = b$. If $[c] = \{c\}$ then the state $c \in A$ is called a *trap* of $\mathbf{A}$. The automaton $\mathbf{A}$ is called *strongly trap-connected* if it has a trap $c$ and for every state $a \in A - \{c\}$ and $b \in A$, there exists a $p \in X^*$ such that $ap = b$. It is known that the automaton $\mathbf{A}$ is strongly connected if and only if it has no subautomaton $\mathbf{A}' = (A', X, \delta)$ of $\mathbf{A}$ such that $A' \neq A$. Furthermore, if $\mathbf{A}$ strongly trap-connected then it has only one trap.

**Corollary 9** (G. Thierrin [16]). *Every simple automaton with at least three states is strongly connected or strongly trap-connected.*

*Proof.* If $\mathbf{A}' = (A', X, \delta')$ is a subautomaton of the automaton $\mathbf{A} = (A, X, \delta)$ then $A'$ is a separator of $\mathbf{A}$. Thus $A' = A$ or $|A'| = 1$. If $\mathbf{A}$ is not strongly connected, then it has only one subautomaton $\mathbf{A}' = (A', X, \delta)$, namely $|A'| = 1$. In the latter case if $A' = \{c\}$ then $c$ is a trap of $\mathbf{A}$. Hence if $\mathbf{A}$ is not strongly connected then it is strongly trap-connected. □

**Theorem 10.** *The strongly trap-connected automaton $\mathbf{A} = (A, X, \delta)$ with at least three states is simple if and only if the trap of $\mathbf{A}$ is disjunctive.*

*Proof.* Let $c \in A$ be the trap of $\mathbf{A}$. First, we show that if $\rho$ is a congruence of $\mathbf{A}$ and $\rho \neq \omega_A$ then $\rho[c] = \{c\}$. Let $a, b \in A$ be arbitrary states. Assume that $(a, c) \in \rho$. If $a \neq c$ then there exists a $p \in X^*$ such that $ap = b$. Thus

$$(b, c) = (ap, cp) \in \rho.$$

From this it follows that $\rho = \omega_A$. This is impossible. Thus we get that $a = c$ and $\rho[c] = \{c\}$.

Now assume that $c$ is disjunctive, that is, $\tau_{\{c\}} = \iota_A$. Let $\rho \neq \omega_A$ be a congruence of $\mathbf{A}$. Since $\rho[c] = \{c\}$, if $a, b \in A - \{c\}$ and $(a, b) \in \rho$ then $(a, b) \in \tau_{\{c\}}$, that is, $a = b$. We get $\rho = \iota_A$ and thus $\mathbf{A}$ is simple.

Conversely, assume that $\mathbf{A}$ is simple. But $\mathbf{A}$ is strongly trap-connected automaton with at least three states, thus $\tau_{\{c\}} \neq \omega_A$. Therefore $\tau_{\{c\}} = \iota_A$ and so $c$ is disjunctive. □

# 4 Commutativity of simple automata

**Theorem 11.** *If the strongly trap-connected automaton $\mathbf{A} = (A, X, \delta)$ with at least three states is simple then it is not commutative. Furthermore $G(\mathbf{A}) = \{\iota_A\}$ and $E(\mathbf{A}) = \{\iota_A, \alpha_c\}$, where $c$ is the trap of $\mathbf{A}$, and $\alpha_c$ defined by $\alpha_c(a) = c$ $(a \in A)$.*

*Proof.* Assume that $\mathbf{A}$ is commutative. Let $a, b \in A - \{c\}$ and $a \neq b$. Since $\mathbf{A}$ is strongly trap-connected, there are $q, r \in X^*$ such that $aq = b$ and $br = a$. Thus, for arbitrary $p \in X^*$,

$$bp = aqp = apq \quad \text{and} \quad ap = brp = bpr.$$

Then, $ap = c$ if and only if $bp = c$. Thus $(a, b) \in \tau_{\{c\}}$, that is, $a = b$, which contradicts the assumption. We get that $\mathbf{A}$ is not commutative.

It is evident that $\alpha_c \in E(\mathbf{A})$. If $\alpha \in E(\mathbf{A})$ then, for every $p \in X^*$,

$$\alpha(c)p = \alpha(cp) = \alpha(c),$$

and so $\alpha(c)$ is a trap of $\mathbf{A}$, that is $\alpha(c) = c$. If $a \in A - \{c\}$ and $\alpha(a) = c$ then, for every $p \in X^*$,

$$\alpha(ap) = \alpha(a)p = cp = c,$$

that is, $\alpha = \alpha_c$. Assume that $a, b \in A - \{c\}$, $a \neq b$ and $\alpha(a) = \alpha(b)$. If, for every $p \in X^*$, $ap = c$ if and only if $bp = c$ then $(a, b) \in \tau_{\{c\}}$. By Theorem 10, $a = b$. This is a contradiction. Thus there exists a $q \in X^*$ such that for instance $aq = c$ and $bq \neq c$. Then

$$\alpha(bq) = \alpha(b)q = \alpha(a)q = \alpha(aq) = \alpha(c) = c.$$

From this it follows that $\alpha = \alpha_c$, thus $G(\mathbf{A}) = \{\iota_A\}$ and $E(\mathbf{A}) = \{\iota_A, \alpha_c\}$. □

**Lemma 12.** *Every endomorphism of a strongly connected automaton is surjective.*

*Proof.* Let $\mathbf{A} = (A, X, \delta)$ be a strongly connected automaton. If $\alpha \in E(\mathbf{A})$ then $\mathbf{A}_\alpha = (\alpha(A), X, \delta')$ is a subautomaton of $\mathbf{A}$. Therefore, $\alpha(A) = A$, that is, $\alpha$ is a surjective mapping. □

**Theorem 13.** *Let the strongly connected automaton $\mathbf{A} = (A, X, \delta)$ with at least three states be simple. If $E(\mathbf{A}) = \{\iota_A\}$ then $\mathbf{A}$ is not commutative. If $E(\mathbf{A}) \neq \{\iota_A\}$ then $\mathbf{A}$ is an $A$-finite commutative automaton, $|E(\mathbf{A}| = |A|$ and $E(\mathbf{A}) = G(\mathbf{A})$ is a cyclic group of prime order.*

*Proof.* First, we show that if the strongly connected automaton $\mathbf{A}$ with at least three states is simple then $E(\mathbf{A}) = G(\mathbf{A})$ is a finite group. Since $\operatorname{Ker}\alpha$ $(\alpha \in E(\mathbf{A}))$ is a congruence of $\mathbf{A}$, $\operatorname{Ker}\alpha = \iota_A$ or $\operatorname{Ker}\alpha = \omega_A$. By Lemma 12, $\alpha$ is surjective

mapping. From this it follows that $\operatorname{Ker}\alpha = \iota_A$ and thus $\alpha \in G(\mathbf{A})$. This means that $E(\mathbf{A}) = G(\mathbf{A})$. By Theorem 3, $E(\mathbf{A})$ is finite.

Assume that $E(\mathbf{A}) = \{\iota_A\}$ and $\mathbf{A}$ is commutative. Since $\mathbf{A}$ is strongly connected, there are $a_0 \in A$ and $p \in X^+$ such that $a_0 \neq a_0 p$. Define the mapping $\alpha_p$ in the same way as in the proof of Lemma 5. Since the relation $\rho_{A,a_0}$ is a congruence on $X^*$, $\alpha_p \in E(\mathbf{A})$ and $\alpha_p \neq \iota_A$. This is impossible, and so $\mathbf{A}$ is not commutative.

Now assume that $E(\mathbf{A}) = G(\mathbf{A}) \neq \{\iota_A\}$. Let $\alpha \in G(\mathbf{A})$ and $\alpha \neq \iota_A$. Consider the congruence $\rho_\alpha$ defined in the proof of Theorem 3. Since $\mathbf{A}$ is simple, $\rho_\alpha = \iota_A$ or $\rho_\alpha = \omega_A$. If $\rho_\alpha = \iota_A$ then $\alpha = \iota_A$. If $\rho_\alpha = \omega_A$ then, for arbitrary state $d \in A$,

$$A = \{d, \alpha(d), \ldots, \alpha^{r-1}(d)\}.$$

If $\beta \in G(\mathbf{A})$ then there exists an integer $0 \leq j \leq r-1$ such that $\beta(d) = \alpha^j(d)$. Thus, for every $p \in X^*$, we have $\beta(dp) = \alpha^j(dp)$, that is, $\beta = \alpha^j$. Then, $G(\mathbf{A}$ is a cyclic group.

If $r$ is not prime then $r = ln$ $(1 < l, n < r)$. Define the binary relation $\rho_{l,n}$ on $A$ as follows. For $a, b \in A$ $(a, b) \in \rho_{l,n}$ if and only if there are integers $0 \leq i \leq l-1$ and $0 \leq j, k \leq n-1$ such that

$$a = \alpha^{i+jl}(d), \quad b = \alpha^{i+kl}(d).$$

It is easy to show that $\rho_{l,n}$ is a congruence of $\mathbf{A}$ and $\rho_{l,n} \neq \iota_A, \omega_A$. It is a contradiction. Hence $r$ is a prime number.

We show that $\mathbf{A}$ is commutative. If $p, q \in X^*$ then let $ap = \alpha^k(a)$ and $aq = \alpha^l(a)$ $(0 \leq k, l \leq r-1)$. Then, for arbitrary $0 \leq i \leq r-1$,

$$\begin{aligned}\alpha^i(a)pq = \alpha^i(ap)q = \alpha^i\alpha^k(a)q = \alpha^i\alpha^k(aq) = \\ = \alpha^i\alpha^k\alpha^l(a) = \alpha^i\alpha^l\alpha^k(a) = \\ = \alpha^i\alpha^l(ap) = \alpha^i\alpha^l(a)p = \alpha^i(aq)p = \alpha^i(a)qp,\end{aligned}$$

that is, $\mathbf{A}$ is commutative.

By Theorem 7, the automaton $\mathbf{A}$ is A-finite. By Lemma 4 and Lemma 5, $|E(\mathbf{A}| = |A|$. □

We note that W. Lex proved in [12], if $\mathbf{A}$ is a simple automaton then $|G(\mathbf{A})| = 1$ or $G(\mathbf{A})$ is a cyclic group of prime order.

The automaton $\mathbf{A} = (A, X, \delta)$ is called a *permutation automaton* if every input sign $x \in X$ is a permutation sign, that is, if $ax = bx$ $(a, b \in A)$ then $a = b$. Let the automaton $\mathbf{A}$ be A-finite and $|A| = r$. The input sign $x \in X$ is called *cyclic permutation sign* if, for any $a \in A$,

$$A = \{a, ax, ax^2, \ldots, ax^{r-1}\} \quad (ax^r = a).$$

The input sign $x \in X$ is called *identical permutation sign* if $ax = a$ for every $a \in A$. The permutation automaton $\mathbf{A}$ is called a *cyclic permutation automaton* of order $r$ if there exists an $x \in X$ cyclic permutation sign.

The congruence $\rho$ of the automaton $\mathbf{A} = (A, X, \delta)$ is called *uniform* if, for every $a, b \in A$, $|\rho[a]| = |\rho[b]|$.

**Lemma 14.** *Every congruence of a strongly connected permutation automaton is uniform.*

*Proof.* Let $\mathbf{A} = (A, X, \delta)$ be a strongly connected permutation automaton. Assume that $\rho$ is a congruence of $\mathbf{A}$ and $a, b \in A$ arbitrary states. Since $\mathbf{A}$ is strongly connected, there are $p, q \in X^*$ such that $b = ap$ and $a = bq$. Then $\rho[a]p \subseteq \rho[b]$ and $\rho[b]q \subseteq \rho[a]$. As every input sign is a permutation sign, we get

$$|\rho[a]| = |\rho[a]p| \leq |\rho[b]| = |\rho[b]q| \leq |\rho[a]|,$$

that is, $|\rho[a]| = |\rho[b]|$. □

From Lemma 14 it follows that every strongly connected permutation automaton of prime order is simple. By the following example this is generally not true.

**Example 15.** If $A = \{1, 2, 3\}$, $X = \{x, y\}$ and

$$1x = 2x = 3,\ 3x = 2,\ 1y = 2,\ 2y = 1,\ 3y = 1,$$

then the automaton $\mathbf{A} = (A, X, \delta)$ is strongly connected of prime order, but not simple.

By the following example, there is a simple strongly connected permutation automaton whose order is not a prime number.

**Example 16.** $A = \{1, 2, 3, 4\}$, $X = \{x, y\}$ and

$$1x = 2,\ 2x = 3,\ 3x = 4,\ 4x = 1,\ 1y = 1,\ 2y = 2,\ 3y = 4,\ 4y = 3.$$

The automaton $\mathbf{A} = (A, X, \delta)$ is a cyclic permutation automaton.

**Theorem 17.** *The commutative automaton $\mathbf{A} = (A, X, \delta)$ with at least three states is simple if and only if it is a cyclic permutation automaton of prime order.*

*Proof.* Assume that the commutative automaton $\mathbf{A}$ is simple. By Theorem 13, $\mathbf{A}$ is an A-finite automaton of prime order. By Corollary 9 and Theorem 11, $\mathbf{A}$ is strongly connected. Let $x \in X$ be an arbitrary input sign. Define the binary relation $\rho_x$ on $A$ as follows.

$$(a, b) \in \rho_x \quad \text{if and only if} \quad ax = bx.$$

Using the commutativity of $\mathbf{A}$, it is not difficult to seen that the relation $\rho_x$ is a congruence of $\mathbf{A}$. If $\rho_x = \omega_A$ then there is an element $c \in A$ such that for every $a \in A$ $ax = c$. Hence $c$ is a trap of $\mathbf{A}$. It is impossible. Thus $\rho_x = \iota_A$, that is, $x$ is a permutation sign. We get that $\mathbf{A}$ is a permutation automaton. Since $\mathbf{A}$ strongly connected and $3 \leq |A|$, there are $a \in A$ and $x \in X$ such that $ax \neq a$. But $x$ is a permutation sign. Therefore, if $ax^i = ax^j$ $(0 \leq i < j)$ then $a = ax^{j-i}$ and $2 \leq j - i$. Let $k$ be the smallest positive integer for which $ax^k = a$. Since $ax \neq a$, therefore $2 \leq k$. The set $H = \{a, ax, \ldots, ax^{k-1}\}$ is a separator of $\mathbf{A}$. From this it follows that $H = A$. Thus $x$ is a cyclic permutation sign, that is, $\mathbf{A}$ is a cyclic permutation automaton of prime order.

Conversely, if $\mathbf{A}$ is a cyclic permutation automaton of prime order then, by Lemma 14, $\mathbf{A}$ is simple. □

If a commutative automaton is a cyclic permutation automaton of prime order then every input sign is an identical permutation sign or a cyclic permutation sign.

We remark that in [16] G. Thierrin proved that if $G(\mathbf{A}) \neq \{\iota_A\}$, for a simple automaton $\mathbf{A}$, then $\mathbf{A}$ is a permutation automaton, $|G(\mathbf{A}| = |A|$ and $|G(\mathbf{A})|$ is a prime number. By Theorem 13, every commutative simple automaton is A-finite. By the following examples, it is generally not true.

**Example 18.** If $A = \{1, 2, \ldots, n, \ldots\}$, $X = \{x, y\}$ and

$$1y = 1,\ 2y = 2,\ nx = n + 1,\ n = 1, 2, \ldots,$$

$$n_1 = 2, \quad n_{i+1} = n_i + i, \quad i = 1, 2, \ldots,$$

$$n_{i+1}y = 1,\ (n_{i+1} + 1)y = (n_{i+1} + 2)y = \cdots = (n_{i+1} + i)y = 2,\ i = 1, 2, \ldots,$$

then the infinite automaton $\mathbf{A} = (A, X, \delta)$ is strongly connected, simple and not commutative.

**Example 19.** If $A = \{0, 1, 2, \ldots, n, \ldots\}$, $X = \{x, y\}$ and

$$0x = 0y = 1y = 0,\ nx = n + 1,\ n = 1, 2, \ldots,$$

$$n_1 = 2, \quad n_{i+1} = n_i + i, \quad i = 1, 2, \ldots,$$

$$n_i y = 1,\ (n_{i+1} + 1)y = (n_{i+1} + 2)y = \cdots = (n_{i+1} + i)y = 2,\ i = 1, 2, \ldots,$$

then the infinite automaton $\mathbf{A} = (A, X, \delta)$ is strongly trap-connected with the trap 0, simple and not commutative.

# References

[1] Babcsányi, I., *A félperfekt kváziautomaták* (*On quasiperfect quasiautomata*), Mat. Lapok, 21 (1970), 95-102 (Hungarian with English summary).

[2] Babcsányi, I., *Rees automaták* (*Rees-automata*), Mat. Lapok, 29 (1977-1981), 139-148 (Hungarian with English summary).

[3] Babcsányi, I., *Simple Mealy and Moore automata*, Proceedings of the International Conference on Automata and Formal Languages IX, Vasszécseny, Hungary, August 9-13, 1999, Publicationes Mathematicae, Supplementum 60 (2002), 473-482.

[4] Babcsányi, I. *Equivalence of Mealy and Moore automata*, Acta Cybernetica 14 (2000), 541-552.

[5] Cohn, P.M., *Universal Algebra*, Harper and Roow Publishers, New York-Evanston-London, 1965.

[6] Ésik Z. and B. Imreh, *Remarks on finite commutative automata*, Acta Cybernetica, 5 (1981), 143-146.

[7] Gécseg, F., *Products of Automata*, Springer-Verlag, Berlin-Heidelberg-New York-Tokyo, 1981.

[8] Gécseg, F. and I. Peák, *Algebraic Theory of Automata*, Akadémiai Kiadó, Budapest, 1972.

[9] Grillet, A.P., *Commutative semigroups with finite congruence lattices*, Acta Sci. Math.(Szeged), 70 (2004), 551-555.

[10] Kim, K.H. and F.W. Roush, *Applied Abstract Algebra*, John Wiley and Sons, New York-Chichester-Brisbane-Ontario, 1983.

[11] Lallement, G., *Semigroups and Combinatorial Applications*, John Wiley and Sons, New York-Chichester-Brisbane-Toronto, 1979.

[12] Lex, W., *Akte* (*Acts*), Habilitationsschift, Clausthal-Zellerfeld, 1980 (German).

[13] Oehmke, R.H., *On the structures of an automaton and its input semigroup*, J. Assoc. Comp. Machinery, 10 (1963), 521-525.

[14] Peák, I., Avtomatü i polugruppü II (*Automata and semigroups II*), Acta Sci. Math.(Szeged), 26 (1965), 49-54 (Russian).

[15] Radeleczki, S., *The automorphism group of unary algebras*, Mathematica Pannonica, 7 (1996), 253-271.

[16] Thierrin, G., *Simple automata*, Kybernetika (Pragua), 5 (1970), 343-350.

*Received February, 2007*

CONTENTS

**Kalmár Workshop on Logic and Computer Science** **1**
Foreword . . . . . 3
László Kalmár (1905–1976) . . . . . 5
*András Hajnal*: In memory of László Kalmár . . . . . 7
*Árpád Makay*: The activities of László Kalmár in the world of information technology . . . . . 9
Publications of László Kalmár . . . . . 15
*Alexander S. Antonenko and Eugene L. Berkovich*: Groups and Semigroups Defined by some Classes of Mealy Automata . . . . . 23
*Edward G. Coffman, Jr. and János Csirik*: A Classification Scheme for Bin Packing Theory . . . . . 47
*Miguel Couceiro and Stephan Foldes*: Functional Equations, Constraints, Definability of Function Classes, and Functions of Boolean Variables . . . . . 61
*Giorgi Japaridze*: Intuitionistic computability logic . . . . . 77

**Regular Papers** **115**
*György Gyurica*: On monotone languages and their characterization by regular expressions . . . . . 117
*Alexander Meduna and Tomáš Masopust*: Self-Regulating Finite Automata . 135
*István Babcsányi*: Automata with Finite Congruence Lattices . . . . . 155

ISSN 0324—721 X

Felelős szerkesztő és kiadó: Csirik János