UC Merced
Proceedings of the Annual Meeting of the Cognitive Science Society

Title
Validating COGNITIO by Simulating a Student Learning to Program in Smalltalk

Permalink
https://escholarship.org/uc/item/9032z055

Journal
Proceedings of the Annual Meeting of the Cognitive Science Society, 14(0)

Authors
Chee, Yam San
Chan, Taizan

Publication Date
1992

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

Validating COGNITIO by Simulating a Student Learning to Program in Smalltalk

Yam San CHEE and Taizan CHAN
Department of Information Systems and Computer Science
National University of Singapore
Lower Kent Ridge Road
Singapore 0511
cheeys@nusdiscs.bitnet
chantz@nusdiscs.bitnet

Abstract

We describe COGNITIO, a computational theory of learning and cognition, and provide evidence of its psychological validity by comparing the protocols of a student learning to program in Smalltalk against a COGNITIO-based computer simulation of the same. COGNITIO is a production system cognitive architecture that accounts parsimoniously for human learning based on three learning mechanisms: *schema formation*, *episodic memory*, and *knowledge compilation*. The results of simulation support the validity of COGNITIO as a computational theory of learning and cognition. We also draw some implications of COGNITIO for the teaching of complex problem solving skills.

Introduction

Existing computational theories of cognition appear unable to account for the complexities of human learning and cognition in a comprehensive and integrated manner (Chan, Chee, & Lim, 1992). The major extant architectures seem incapable of providing us with a suitable framework for modeling students learning to program in Smalltalk.

Although Anderson's ACT* and PUPS theories (Anderson, 1983; Anderson, 1989) are able to account, in detail, for skill acquisition, they are unable to describe other aspects of learning such as *assimilation* of new declarative domain knowledge, reliance on prior problem solving *episodes*, and the *formation* of memory *schemata*. Similarly, SOAR (Laird, Rosenbloom, & Newell, 1986; Laird, Newell, & Rosenbloom, 1987) is able to account for skill acquisition but shares the same deficiencies of ACT*.

Other computational theories such as Schank's MOPs and TOPs (Schank, 1982) and Kolodner's E-MOPs (Kolodner, 1983; Kolodner, 1987), on the other hand, account for cognition only from a restricted point of view, namely, that of case-based reasoning (see Slade (1991) for a discussion of the case-based reasoning paradigm). Case-based reasoning and problem solving based on episodic memory are important elements of human cognition that ACT* and SOAR do not account for. Unfortunately, case-based reasoning, on its own, is unable to account for the finer problem solving and learning behavior that ACT* and SOAR offer through their production system architecture and their knowledge compilation (Neves & Anderson, 1981; Anderson, 1987) and chunking mechanism (Laird, Rosenbloom, & Newell, 1986) respectively.

Theories such as Induction (Holland, Holyoak, Nisbett, & Thagard, 1987) and Repair Theory (Brown & Vanlehn, 1980) are also problematic from the perspective of accounting for complex problem solving behavior. The theory of Induction accounts for learning in a problem solving domain through activation: a rule which has been successfully applied will be more highly activated than another which has been unsuccessfully applied. Consequently, when a similar problem arises the next time, the more highly activated rule will be selected first. The theory of Induction seems to describe cognition at a level which is too low for modeling complex problem solving behavior. Furthermore, it omits the role of schematic knowledge in problem solving, an important indication of increasing expertise in a domain (Chi, Feltovich, & Glaser, 1981; Rumelhart, 1980; Rumelhart & Norman, 1978). Repair theory, on the other hand, has been most successfully applied to the study of subtraction. However, it is difficult to extend it to the modeling of programming behavior. While modeling two or three column subtraction can be achieved using a very small number of operators, the programming process entails a much more complex sequence of planning and reasoning steps. Unlike subtraction, the range of possible impasses is virtually unbounded in programming. In addition, Repair Theory also excludes the phenomenon of schematic memory organization. Consequently,

Repair Theory is ill-suited to studying programming behavior.

In response to the perceived deficiencies of existing cognitive theories, we have formulated COGNITIO as an integrated theory of learning and cognition. COGNITIO is particularly well-suited to accounting for learning and cognition by incorporating the formation of schematic knowledge, the use of episodic memory, and the compilation of knowledge.

COGNITIO: An Extended Theory of Cognition

COGNITIO is essentially an extension to the ACT* theory. The architecture of cognition embodied in COGNITIO is given in Fig. 1.

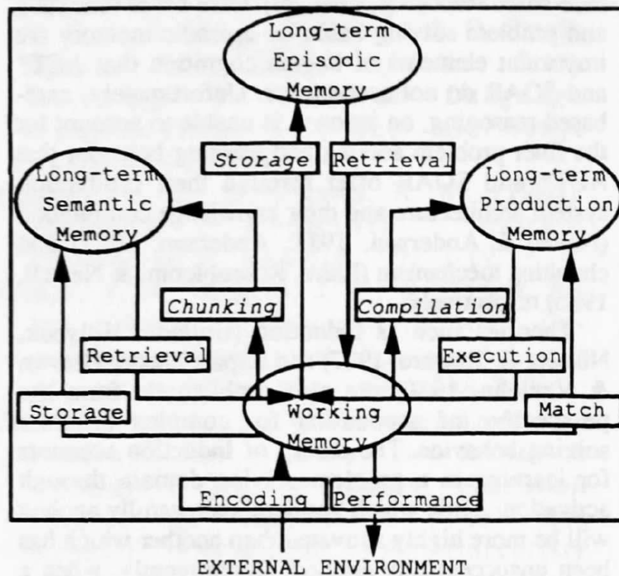

Fig.1 The Cognitive Architecture *COGNITIO*

COGNITIO is a production system theory of cognition. It contains a *working memory* and three separate long-term memories – *semantic memory*, *episodic memory*, and *production memory*. Working memory is a short-term, limited capacity memory which is the activated portion of the semantic memory. Episodic memory contains prior problem solving episodes which are represented as plan-trees (Chan, Chee, & Lim, 1992). When the conditions in a production rule contained in the long-term production memory match the state of the working memory, the rule is fired, and the actions (mental and possibly physical) are performed. Cognitive behavior is the result of a series of production matchings and firings.

More importantly, COGNITIO is also a theory of *learning*. COGNITIO postulates that the long-term memories are transformed *as a result of production firings*. That is, learning in a domain occurs only if that domain knowledge is used in solving problems of the domain. These long-term memory transformations are evidenced by increasing competence in a domain. The transformations occur through three learning mechanisms: *schema formation*, *episode storage/retrieval*, and *knowledge compilation*.

(i) *Schema Formation* (or Declarative Chunking). Related memory elements from semantic memory which are often accessed together in solving problems are organized together into higher-level memory structures – *schemata* – which can be accessed later as individual units in working memory. Schema formation reduces the demand on working memory and enables a person to view a complex concept or problem in an appropriate context. The degree of coherence and understanding achieved by virtue of a newly acquired schema will depend on how well elaborated it is. In addition, a schema that has become a unit by itself can become part of a larger schema as the learner continues with his learning. Such schema formation corresponds to the ability to build a better mental model of a problem at hand since more information is made available, in a coherent form, at the time a person reasons about the problem.

(ii) *Episode Storage/Retrieval*. A person can rely on his prior experience to guide the solution to a new but similar problem. Each problem-solving experience is considered an *episode* and is stored in the long-term *episodic memory*. When a new problem bears some resemblance to a problem that has been solved previously, a person could be reminded of that previous problem-solving episode. He would then rely on that episode for some of the steps performed or decisions made previously, instead of attempting to solve the current problem anew. An episode is retrieved when an episode's goal and conditions match the current goal and current working memory content.

(iii) *Knowledge Compilation*. Knowledge compilation accounts for increasing fluency in skill acquisition and ultimately to a high degree of automaticity. The two submechanisms of knowledge compilation are *proceduralization* and *composition*. Proceduralization creates a new production by removing or modifying conditions that require access to long-term semantic memory so that the semantic knowledge is built into the new production itself. Composition creates new productions by composing two productions that are fired in sequence in achieving related goals into another production that, when fired, will have the same effect as the two original productions fired in sequence.

In summary, COGNITIO is a computational theory of learning and cognition. It improves upon earlier models of cognition by incorporating the role of

episodic memory in a production-based architecture and by specifically including a mechanism for schema formation and accommodating the role of schemata in domain knowledge assimilation.

Validating COGNITIO

This section describes a COGNITIO-based simulation of a student's learning and problem-solving behavior. The first subsection describes the nature of the protocols of a student learning to program in Smalltalk. These protocols provide the psychological data for testing the validity of COGNITIO. The second subsection describes the simulation itself. The third subsection summarizes the simulation results.

Protocol Collection

We performed a simulation of a student learning to program in Smalltalk based on the protocols collected from a first-year computer science undergraduate at this university. The student had only studied Pascal programming for one semester. Both video and audio protocols were collected of the entire Smalltalk study session (total duration of approximately six hours). The student read aloud the instruction and verbalized his thoughts while trying to understand the instructions and while solving problems. The instructions explained Smalltalk concepts and the programming interface. They also required the student to solve problems periodically by being engaged in a financial game. An instructor was present to provide assistance whenever necessary. The video recordings captured the interaction between the student and the Smalltalk programming environment. The audio recordings captured the verbalizations of the student and the dialogs between the student and the instructor.

The Simulation

This section describes the simulation of the opening segment of the protocols. A schematic version of the protocols is contained in the Appendix. The first subsection describes our approach to simulating the learning embodied in the protocols while the second subsection describes the simulation in detail.

Process of simulation. A computer system based on the COGNITIO architecture and theory was implemented to simulate the student's protocols. The process of simulation runs directly parallel to the unfolding learning behavior. Consequently, at points where the student read instruction, knowledge corresponding to the instruction was encoded (in propositional form) and entered into the system. Similarly, at points where a problem had to be solved, the system was given the corresponding problem solving goal to achieve. Thus, a detailed, step-by-step simulation of the learning process was performed. Such an approach is significant in that knowledge acquired at time *t1* affects not only behavior at time *t2* (later) but also how new knowledge entered into the system at time *t2* is interpreted and integrated into the system memory. Consequently, assimilation of new declarative knowledge is directly dependent on what is already known.

The simulation results. At the commencement of the simulation, the system only contains a few productions that represent the weak problem solving method of *divide-and-conquer*. These productions, like other productions that are used for modeling problem-solving behavior, have actions that would normally generate subgoals to achieve a given goal. For example, one weak-method production encodes the following rule:

> **If** a goal is to achieve a certain function and
> some operation can achieve that function and
> that operation has a number of steps
> **then** set subgoals to perform those steps.

The long-term semantic memory also contains the propositional form of the instruction that was read by the student before he attempted the first problem, namely, to determine the balance in the account object `MyAccount`. Like the student (in steps 4 to 8), the system solves the problem in the three steps given, namely, by typing the expression "`MyAccount` **`queryBalance`**", highlighting the expression, and selecting "print it" from the operate menu. The steps were generated as three subgoals in the simulation. After solving this problem, no schema is formed and no productions are compiled because the level of activation of semantic memory elements involved in solving the problem does not exceed a prespecified threshold. However, the system's episodic memory has been registered with the steps the system took to evaluate the expression "`MyAccount` **`queryBalance`**".

The preceding episode guides the solution to the second problem of determining the interest rate of `MyAccount`. The system, like the student (in steps 10 to 12), is able to type out the expression that was thought to be needed, to highlight it, and to select "print it". This behavior is best explained as behavior that results from invoking the knowledge stored from the first problem solving episode. It cannot be explained in terms of knowledge compilation because the student is, at this point, still verbalizing the need to highlight the expression (*"so we select the whole thing"*) and no production was compiled after the first problem solving episode.

The system, like the subject (in step 10), however, fails to type the object `MyAccount` in the

expression evaluated. There are two reasons for this. First, the previous episode only encoded what steps needed to be performed to achieve the desired function, namely, to type the expression, highlight it, and select "print it". The episode did not encode the knowledge about the structure of a message expression – that a message expression consists of an object followed by a message. Secondly, there was a lack of explicit specification of the object in the instructions but strong association of **interestRate** to the function of determining interest rate. As a result, the expression to be evaluated is associated with only the message **interestRate** in the working memory leading to the incorrect behavior observed. However, when the cause of the error was pointed out by the instructor (missing receiver object, MyAccount, in the message expression), the student (in step 14) was able to make the appropriate correction (steps 15-17).

When the system (and the student in step 18) is given the goal to start up the trading place, it (like the student) is able to code the expression correctly (steps 19-21). Furthermore, from the verbalizations of the student (*"TradingPlace is the object"* and *"**startUp** is the message"*), it can be seen that a schema encoding the structure of a message expression is being formed. The system simulates this in corresponding fashion by creating a *message expression* schema that comprises an object slot, a message slot, and a constraint slot specifying that the receiver object must precede the message selector.

As the simulation proceeds, the message expression schema is enhanced with a slot indicating that a Smalltalk object is returned from the evaluation of a message expression (steps 23-25). Evidence of this becomes apparent later (step 31) by the way the student interprets the nested message expressions in step 30. Furthermore, a production that proceduralizes the steps to evaluate a message expression is also formed. The omission of verbalization and the speed with which the expression was evaluated (that is, highlighting and selecting "print it" in step 23) are supportive of the proceduralization having occurred.

As the student progressed with his learning, he acquired more schemata, more experience from prior problem-solving episodes, and more productions formed in production memory. For example, after about an hour and a half after the protocol extract shown in the Appendix, the student had also formed an appropriate *method* schema that resulted from coding several expression series, examining and understanding methods in detail, and coding a method. Supporting evidence for this schema formation is provided when the student was trying to understand the following given method:

```
buyShareOfName: aString
 SharesPortfolio add:(TradingPlace
          buyShareOfName:aString
          usingAccount: self)
```

in which the global variable SharesPortfolio is accessed in the method. The student elaborated his *method* schema in the following verbalization: *"SharesPortfolio being a global variable, we can do this"*. This is because the method schema specifies the parts of a method as well as the constraint that only instance and class variables are accessible in the method of the receiver. Thus, upon encountering Sharesportfolio which is neither the instance or class variable of an account object, the student is trying to interpret the method in a manner which is consistent with his current schema.

Evidence for problem solving based on episodic memory is provided at a point in time after the student coded the above method. The student remembered that he had previously used another message which drew cash directly from the account to get a higher discount: *"remember there is **usingCashOf**"*, (after locating the actual message name) *"so, if I were to rewrite my method...of the...to **usingCashOf withdraw** amount, wouldn't that be better?"*. As a result, he changed the given method to the one that gave a higher discount. This behavior can be simulated by inserting an additional condition in the working memory that a maximum discount is required so that the episode involving the use of the nested message:

```
"TradingPlace buyShareOfName:aShareName
              usingCashOf:(MyAccount
                  withdraw:anAmount)"
```

is retrieved. Without the learning mechanism of episodic memory, the working memory must first be augmented with the various facts and relationships between the messages **withdraw:** and **buyShareOfName:usingCashOf:**. A series of reasoning steps must then be performed to construct the appropriate nested message. However, the protocol of the student did not reveal the occurrence of such reasoning steps. In addition, knowledge compilation cannot be used to explain the behavior because the student was unable to reproduce the whole nested expression readily. Instead, he was reminded of the episode, but still had to rely on the text for the exact form of the message expression required.

There was also ample evidence of knowledge compilation. However, most of the rules compiled were interface-related; for example, copying a method from one class to another through the system browser. Since the knowledge compilation mechanism operates the same way as in ACT*, we do not elaborate on this further.

Simulation Conclusions

The simulation results described above demonstrate that COGNITIO provides a faithful computational account of cognition and learning. In particular:

(i) The basic production system architecture can be used to simulate cognition, and especially learning. This is consistent with the ability of ACT* (Anderson, Farrell, & Sauers, 1984) and SOAR (Lewis, et al, 1990) to simulate human problem solving behavior.

(ii) The three learning mechanisms, namely, schema formation, storage/retrieval of episodes, and knowledge compilation, as embodied in COGNITIO are all essential components of any account of human learning. Taken individually, each mechanism may account for one aspect of learning but not another. By integrating the three learning mechanisms in a parsimonious way, COGNITIO is able to account for learning in a more integrated and powerful way.

Implications of COGNITIO

The simulation highlights some implications for teaching Smalltalk and other skills in general:

(i) It is important that a student be given ample practice and exposure to concepts (for example, the structure of message expressions) before he has to proceed further in a course of instruction. This is to ensure that appropriate knowledge has been schematized so that it will leave more working memory capacity available for new concepts to be acquired.

(ii) Instruction for teaching a skill, such as Smalltalk programming, should be designed with the aim of equipping the student with the various schemata of the essential concepts of the skill. For example, in Smalltalk, some relevant schemata are the message expression schema, the method schema, the class schema, and so on. The observation is that a schema can aid the student in understanding related concepts that will be introduced later. For example, a schema for message expressions is essential for understanding more complicated concepts such as nested message expressions or methods.

(iii) Parts of a knowledge or skill can be acquired through episodic learning but other parts might be better taught explicitly. This is illustrated by the earliest part of the simulation when the student had no difficulty performing the steps to evaluate an expression but forgot to include the object in the message expression. Thus, a "semantic" relationship, such as that between an object and a message, is best taught explicitly; failure to do so entails the risk of the relationship being ignored by a student.

Conclusions

This paper has described a computational theory of learning and cognition, as embodied in COGNITIO. The theory is able to account for the schematic organization of memory, the role of episodic memory in learning, and skill acquisition in cognition. Evidence of the validity of COGNITIO was demonstrated by its application to the simulation of a student learning to program in Smalltalk. The simulation also highlights some implications for teaching problem-solving skills such as Smalltalk programming.

Appendix. A schematic protocol of a student learning to program in Smalltalk is shown below. The student's actual verbalizations are shown in italics in square brackets.

1. Student learns that to achieve something in Smalltalk, he must send a message to some object.
2. Student learns that the basic format to do so is "object message-name".

[Problem Solving Episode 1: Steps 3-8]

3. Student is asked to determine the balance in `MyAccount` by sending the message **`queryBalance`** to the object `MyAccount` in the Workspace window (already opened).
4. The steps given are: (a) type "`MyAccount` **`queryBalance`**", (b) highlight the expression, and (c) select "print it" from the operate menu.
5. Student types the expression.
6. Student highlights the expression.
7. Instructor teaches how and where to select "print it"
8. Expression is evaluated – 5000 (which is the current balance in `MyAccount`) is shown beside the evaluated expression.

[Problem Solving Episode 2: Steps 9-12]

9. Student is asked to determine the interest rate offered in `MyAccount` by sending it the message **`interestRate`**.
0. Student thinks aloud [*"**interestRate**"...*] as he types "**`interestRate`**"
11. Student thinks aloud [*"so we select the whole thing"...*] as he highlights "**`interestRate`**"
12. Student selects "print it".
13. Error message appears (because the system treats **`interestRate`** as an object according to the syntax of Smalltalk, and such an object does not exist).
14. Instructor points out the mistake that the object in the expression has been omitted.

[Problem Solving Episode 3: Steps 15-17]

15. Student types "`MyAccount`" before "**`interestRate`**".
16. Student thinks aloud [*"**interestRate** is not object, **interestRate** is message"*].
17. Student evaluates the expression – 0.04, which is the interest rate, is shown.

[Problem Solving Episode 4: Steps 18-21]

18. Student is asked to start a `TradingPlace` running by sending it a message, **`startUp`**.
19. [*"`TradingPlace` is the object"*] types "`TradingPlace`".
20. [*"**startUp** is the message"*] types "**`startUp`**".
21. Student evaluates the expression by highlighting it and selecting "print it" (a window representing a trading place appears).

[Problem Solving Episode 5: Steps 22-25]

22. Student is asked to buy shares by sending the message **buyShare** to TradingPlace.
23. Student types "TradingPlace", and then "**buyShare**"; he then evaluates it – anUserShare is shown. [In Smalltalk, an object is always returned as a result of executing a message expression. If a user selects "print it" as the command for executing an expression, the object will be printed. If an object is not a literal, its class will be printed.]
24. Student tries to determine what is anUserShare.
25. Instructor points out that it is an indication that a share object is returned.
[Problem Solving Episode 6: Steps 26-27]
26. Student is asked to buy shares by evaluating the expression
"TradingPlace **buyShareOfName**:'Emtex'
usingAccount:MyAccount".
27. Student types "TradingPlace", then types "**buyShareOfName**:'Emtex'" and then types "**usingAccount**:MyAccount" and then evaluates it – anUserShare is shown.
28. Student is told that the he has lost access to the shares bought because they were not stored away somewhere.
29. Student is told that the shares to be bought later can be stored in the set object SharesPortfolio and that a share can be added into the set by sending it the message "**add**:aShare".
30. Student is asked to evaluate one of the messages in order to buy shares and add it into the SharesPortfolio.
(i) SharesPortfolio **add**:TradingPlace
buyShare
(ii) SharesPortfolio **add**:
(TradingPlace **buyShareOfName**:shareName
usingAccount:anAccount)
(iii) SharesPortfolio **add**:
(TradingPlace
buyShareOfName:shareName
usingCashOf:
(anAccount **withdraw**:amount))
31. Student interprets the above messages (i) and (ii) as [*"**add:** is the message...message a share...the argument being sent in is...when is returned is anUserShare and this is an object so goes into **add:** and this **add:** goes into SharesPortfolio which is the set object"*] and then (iii) as [*"**withdraw:** is another message whereby it allows you to say how much to withdraw ...and same thing...and account withdraw will return you an object..."*]

References

Anderson, J. R. 1983. *The Architecture of Cognition*. Cambridge, MA: Harvard University Press.

Anderson, J.R. 1987. Skill acquisition: Compilation of weak-method problem solutions. *Psychological Review*, 94(2): 192-210.

Anderson, J.R. 1989. A theory of the origins of human knowledge. *Artificial Intelligence*, 40: 313-351.

Anderson, J. R., Farrell, R., and Sauers, R. 1984. Learning to program in Lisp. *Cognitive Science*, 8: 87-129.

Brown, J. S., and Vanlehn, K. 1980. Repair theory: A generative theory of bugs in procedural skills. *Cognitive Science*, 4: 379-426.

Chan, T., Chee, Y. S., & Lim, E. L. 1992. COGNITIO: An Extended Computational Theory of Cognition. *Proceedings of ITS'92*. New York, NY: Springer-Verlag. Forthcoming.

Chi, M. T. H., Feltovich, P., and Glaser, R. 1981. Categorization and representation of physics problems by experts and novices. *Cognitive Science*, 5: 121-152.

Holland, J. H., Holyoak, K. J., Nisbett, R. E., and Thagard, P. R. 1987. *Induction: Processes of Inference, Learning, and Discovery*. Cambridge, MA: MIT Press.

Kolodner, J. L. 1983. Towards an understanding of the role of experience in the evolution from novice to expert. *International Journal of Man-Machine Studies*, 19: 497-518.

Kolodner, J. L. 1987. Extending problem solving capabilities through case-based inference. In *Proceedings of Forth Annual International Learning Workshop*. Los Altos, CA: Morgan-Kaufmann.

Laird, J. E., Rosenbloom, P. S., and Newell, A. 1986. Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning*, 1: 11-46.

Laird, J. E., Newell, A., and Rosenbloom, P. S. 1987. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33: 1-64.

Lewis, R. L., Huffman, S. B., John, B. E., Laird, J. E., Lehman, J. F., Newell, A., Rosenbloom, P. S., Simon, T., and Tessler, S. G. 1990. Soar as a unified theory of cognition. In *Proceedings of the 12th Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Lawrence Erlbaum.

Neves, D. M., and Anderson, J. R. 1981. Knowledge compilation: Mechanisms for the automatization of cognitive skills. In J.R. Anderson (ed.), *Cognitive Skills and Their Acquisition*. Hillsdale, NJ: Lawrence Erlbaum.

Rumelhart, D. E. 1980. Schemata: The building blocks of cognition. In R. J. Spiro, B. C. Bruce, and W. F. Brewer (eds.), *Theoretical Issues in Reading Comprehension*. Hillsdale, NJ: Lawrence Erlbaum.

Rumelhart, D.E., and Norman, D. A. 1978. Accretion, tuning, and restructuring: Three modes of learning. In J. W. Cotton and R. L. Klatzky (eds.), *Semantic Factors in Cognition*. Hillsdale, NJ: Lawrence Erlbaum.

Schank, R. C. 1982. *Dynamic Memory – A Theory of Reminding and Learning in Computers and People*. Cambridge: Cambridge University Press.

Slade, S. 1991. Case-based reasoning: A research paradigm. *AI Magazine*, 12(1): 42-55.