Tapping into the Language System with Artificial Languages when Learning Programming

ACM Reference Format:
. 2023. Tapping into the Language System with Artificial Languages when Learning Programming. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 BROCANTO

In this section, we provide an overview of *Brocanto*, its use cases, and how we used it in our study. We start by discussing why it is a suitable artificial language in the context of programming learning.

1.1 Motivation

In experiments on natural languages, artificial languages have proven to be valid substitutes for natural languages [6, 7]. As we put the natural language (learning) into focus in this study, we decided against a treatment with another programming language. Therefore, it would have been a study on transfer. For the same reason, we also did not choose to contradict syntax exercises with Python, so called 'Drill tasks' as we wanted to implicitly activate learning mechanisms that are used in natural language learning. Our goal is to evaluate whether learning an artificial language can act as a transfer medium of concepts from a natural language to a programming language. This way, we tap into the language system to make programming learning easier. It is reasonable to believe that this is possible, because artificial grammars, much like programming languages, have strict structure rules and are typically fast to learn (i.e., within a few hours) [3, 4, 6].

Brocanto consists of universal principles of natural languages, such as (pseudo)words, phrases, and syntax rules, which also count for programming languages. Thus, *Brocanto* can act as a bridge element between natural languages and programming languages. Additionally, *Brocanto* supports transfer, so that knowledge from a known natural language is applied in learning *Brocanto* [11]. This can be seen in phrase structure rules and finite-state grammar, which are language learning methods that are active in the acquisition of an artificial language [8, 9].

First, the phrase structure rules method describes that language learning entails breaking down a language's constituent parts, also known as syntactic categories, into its phrases. Since *Brocanto* has a phrase structure grammar, a transfer of this knowledge of natural languages can be applied to learning *Brocanto*. In other words, when learning *Brocanto*, students decompose *Brocanto* sentences in substructures as part of the learning process, which also happens during processing natural languages [8, 9].

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Second, the finite state grammar focuses on the probability of transition of individual elements, which increases with an increasing similarity between languages. For example, Brooks and Vokey found that, with similarity-based learning, students apply grammatical knowledge to a new set of letters (from artificial grammars) [2]. Thus, what is similar is also more easily transferred between languages. Both learning strategies have been observed in studies with *Brocanto*, making it particularly suitable for our experiment.

1.2 Definition

With its different nominal and verb phrase formations, *Brocanto* allows the definition of main clauses with a subject-verb-[object] structure. Formally, *Brocanto* is defined as:

- Start symbol S
- S := NP ∘ VP | NP ∘ VP ∘ NP ▷ *Sentence*
- NP := d ∘ N | D ∘ M ∘ N ▷ *Nominal Phrase*
- VP := v | v ∘ m ▷ *Verb Phrase*
- N := gum | trul | plox | tok ▷ *Noun*
- v := pel | prez | glif | rix ▷*Verb*
- M := böke | füne ▷*Adjective*
- m := nöri | rüfi ▷*Adverb, Verb-Suffix (Conjugation)*
- d := aaf ▷*Definite Article, Pronoun*
- D := aak ▷*Definite Article, Pronoun*

The nominal phrase (NP) consists of a determiner (D, d), optionally an adjective (M), and a noun (N). Thus, a nominal phrase can be written formally as (dN or DMN). Verb phrases (VP) consist of a verb (v), and optionally an adverb or verb-suffix (m), and can be written as v or vm. Thus, the sentence aaf plox prez aak böke trul is correct according to *Brocanto* syntax, but aaf plox prez böke trul is not, because prez böke trul does not follow the construction of nominal phrases, as böke trul does not have a determiner (aaf or aak).

The standard protocol to teach artificial languages is to confront participants with grammatically correct and incorrect sentences [3, 8, 9]. There are three different types of grammatical violations that have proved successful for learning *Brocanto* [5, 7]: Phrase order violations, determiner-noun agreement violations, and word category repetitions.

Phrase order violation. The valid phrase order of a sentence is violated when the order of phrases does not conform to NP+VP or NP+VP+NP. For example, if the verb phrase is put at the beginning of a sentence, the phrase order is violated (see Table 1 for an example).

Determiner-noun agreement violation. Brocanto distinguishes between two determiners: one that can form the nominal phrase with the noun alone (d) and another that can form the nominal phrase only with the addition of an adjective (D+M). The determiner D must be followed by an adjective M before the noun N. The determiner d can only be followed by a noun N. Ignoring this rule and

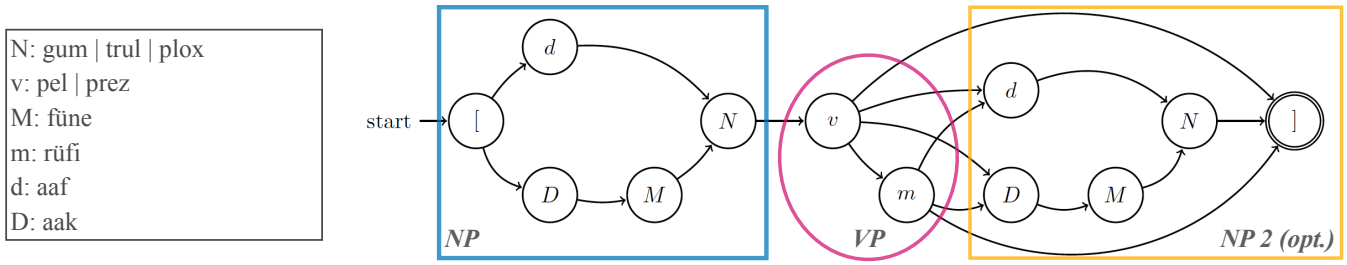

Figure 1: *Brocanto* Transition Diagram / Automat

Table 1: Violation Overview

Violation	Structure Example	Brocanto Example	Natural Language Example
No Violation	NP +VP +NP	aak füne trul prez aaf plox	My black cat makes the noise.
Phrase order violations	**NP** + **NP** + VP	aak füne trul **aaf plox** prez	My black cat **the noise** makes.
Determiner-noun agreement violations	**d**+**M**+N+v+d+N	**aaf füne** trul prez aaf plox	**My the** black cat makes the noise.
Word category repetitions	D+**N**+M+**N**+v+d+N	aak **plox** füne trul prez aaf plox	My **noise** black **cat** makes the noise.

adding an adjective M to the determiner d leads to an agreement violation (cf. Table 1).

Word category repetition. Since *Brocanto* has fixed word categories, each word can be assigned a concrete position within a phrase; within phrases, word categories cannot repeat. If two nouns occur in one nominal phrase, the structure of that phrase is violated; it does not matter at which position the word category repetition is placed (cf. Table 1).

We use these three kinds of violations in our experiment.

1.3 Adaptation

For our experiment, we adapted *Brocanto* to fit within the time frame of our study, as the full version takes several hours to learn. The stimulus material that we created followed the treatment according to Opitz 2011 [5] included all pseudowords from *Brocanto*. The pilot showed that in the context of our short treatment, using sentences with all possible 14 pseudowords leads to cognitive overload. So we adapted the material to Opitz [6], in which the procedure more closely matched our time period. Specifically, we restricted the sentence length to 5 to 8 words and shortened the set of *Brocanto*'s pseudowords. From the 14 pseudowords, we included 9 in our study (3xN, 2xv, 1xM, 1xm, 1xd, 1xD; cf. Fig. 1).

2 EXPERIMENT DESIGN

Having introduced *Brocanto*, we present the design of our study. Details of questionnaires, course material, and data are available at the project's Web site (https://anonymous.4open.science/r/BROCAtreat).

2.1 Task and Material

This section explains the questionnaires, tests, and material in the order in which they appeared in our study, which was run over one week.

Pretest. The first day of the study included an introduction round to welcome participants and to assess previous programming experience. Then, students completed an adapted version of an established programming experience questionnaire [10] and a test to assess existing programming skills [1]. The pretest by Ahadi and others provided us with an extended assessment of students' experience by using an applied variable swap to evaluate whether basic programming concepts are already known. It consists of 8 tasks, each of which can either be solved correctly (1 point), or incorrectly (0 points), leading to a maximum of 8 points in this test. The results from the questionnaire and pretest were the basis to create two comparable groups regarding programming skill. On the next day, the two groups individually received their treatment in *Brocanto* or an introduction to Git.

Treatment Brocanto. For the treatment, we created 60 correct sentences and 60 incorrect sentences, 20 of each of the three types of violation (cf. Section 1.2). For teaching *Brocanto*, we followed standard procedure with three types of blocks that were repeated: Learning, testing, and distractor (see Figure 1 for an overview) [3, 4, 7]. In the *learning block*, participants see 10 correct sentences, each for 7 seconds, and are instructed to deduct the underlying grammar rules. Between each of the 10 sentences, a fixation cross was shown for 2 seconds in the center of the screen.

In the *test block*, participants see 10 sentences, each for 7 seconds. 5 of the sentences are correct, and 5 incorrect, containing any of the three violations, (cf. Section 1), in random order. Of the correct sentences, around half were already shown in the learning block, and the other half were entirely new. The sentences with violations were new to the participants, and no violated versions of the sentences shown in the learning block. Due to *Brocanto's* limited repertoire of words and phrases, sentences that were shown in correct form in the *learning block* may appear in violated form in

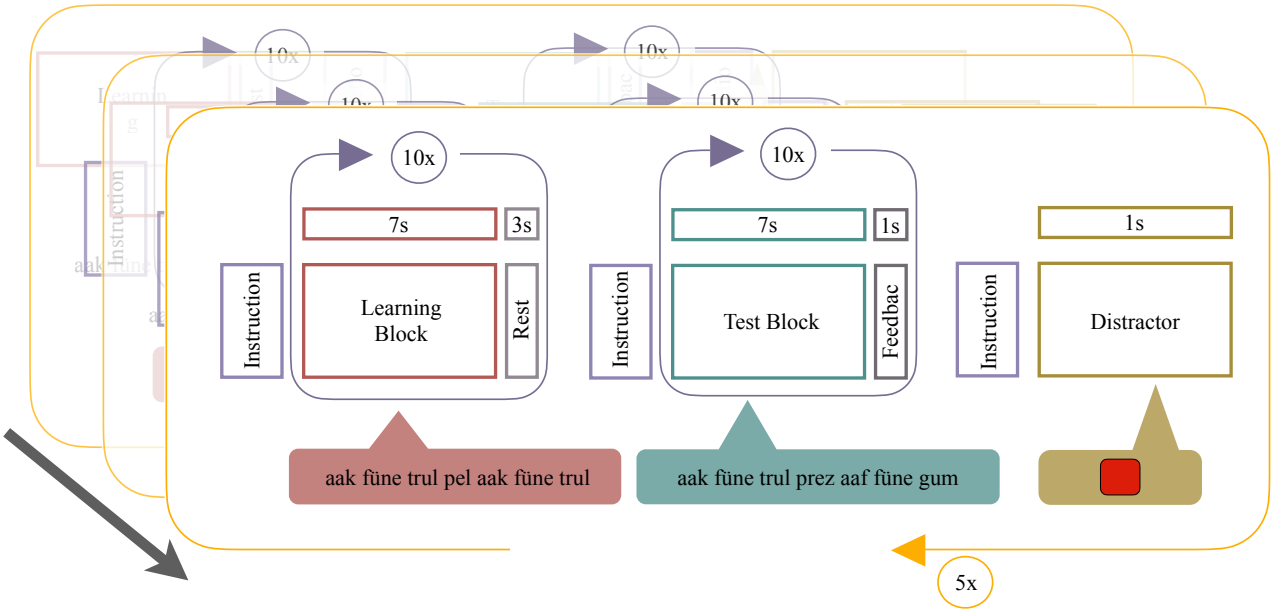

Figure 2: Illustration of one experiment cycle

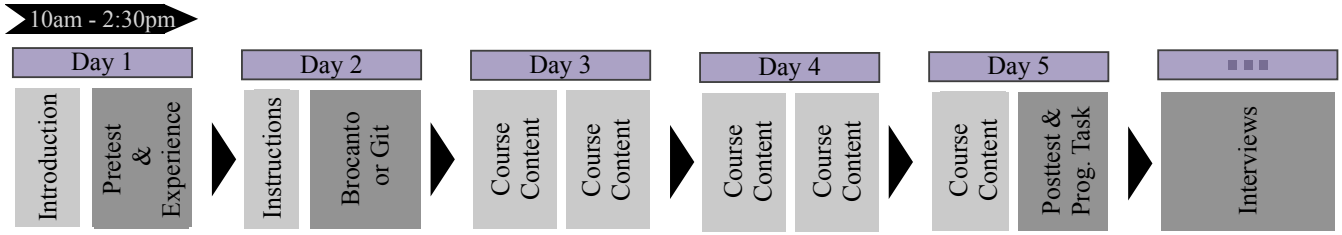

Figure 3: Procedure of Programming Course

one of the *test blocks*. Because of the violations, the sentences are not comparable. Participants were instructed to intuitively decide as fast as possible whether the sentence was correct or incorrect by pressing the right (correct) or left (incorrect) arrow key. After each decision, participants immediately received feedback for 1 second. If participants did not respond within the 7 seconds, the decision was logged as incorrect and participants could proceed by pressing the space bar.

The purpose of the *distractor block* was to inhibit memorization of grammar rules and correct/incorrect sentences, so participants still deduct them in subsequent blocks. To this end, the *distractor block* contained a forced key choice, in which participants saw one of two colored geometrical shapes. For a red square, participants should press *y*, followed by *space*, and for a yellow circle, *n*, followed by *space*.

These three blocks were repeated five times, summarized into one cycle. The experiment consisted of 3 cycles. After each cycle, participants could take a short break (e.g., loosen their hands) and continue by pressing the space bar. The two geometrical shapes that participants had to react to in the *distractor block* did not change. Since it is easy to remember the two key choices, the *distractor block* might no longer distract properly towards the end of the experiment. For this reason, the instruction on how to react was omitted in the last 3 cycles. The participants had to react from memory with the correct key choice when seeing the respective geometric shape. They were not informed beforehand that the instruction would be omitted at the end. The active confusion was to support that they had to fall back on implicit learning that was trained in the experiment in the last *test blocks*.

Participants completed a short warm-up cycle of the same setting consisting of 4 combinations of an artificial grammar of letters to familiarize themselves with the experiment setting.

Control Git. Participants received a brief introduction to version control systems and the advantages of Git over clouds or local version control systems. Afterward, we taught the basic concepts and commands of Git (e.g., push, pull, commit). We showed them how to create new projects, add content, and create new branches with GitHub Desktop. During the creation of the new branches, we explained potential merge conflicts. After this introduction, the participants were able to create a new project, clone a project, and

use the standard Git-commands. The participants did not practice programming commands in the process.

Programming Instructions. The remaining days, both groups met again in the same room and received programming training in Python, covering basic aspects, such as variables, data types, and control structures, adapted from a course of Xie and others [12]. This included intermediate programming assignments to asses whether the concepts were understood.

Posttest. On the last day of the programming course, a posttest was administered. It consisted of two parts: First, the pretest was repeated, but with different variable values. Second, participants solved a programming task, which was to compute the average of all numbers between 1 and 100 that were multiples of 5 and print the result. The solutions were categorized into correct (2 points), conceptually mostly correct (1 point), and incorrect (0 points).

Interviews. Within two weeks of the study, we conducted retrospective voluntary interviews with 4 participants from the course. Of these, 3 belonged to the *Brocanto* group and 1 to the Git group. With the interviews, we gain detailed insights into possible applied strategies of the students in learning the artificial language as well as programming by having them reflect on the experiment, the Git introduction, and the course assignments. The interviews lasted about 30 minutes. The participants received a small expense allowance. The two course instructors each interviewed one participant together in a semi-structured interview. In general, participants were encouraged to reflect freely on topics. Open questions about possible problems in programming, repeating the tasks in the experiment and Git, and whether the treatment was helpful or hindering for learning programming provided the frames for reflection. The interview was digitally recorded and the recording was transcribed and anonymized in a further step for data protection and analysis reasons.

2.2 Procedure

We show an overview of the study in Figure 3. All 20 participants came to the same room, were greeted, and the details of the course were explained, after which the pretest and questionnaire were administered. On the next day, the *Brocanto* group and Git group met in different rooms and learned *Brocanto* or Git. The remaining days covered the introduction to programming, and the last day concluded with the posttest, after which the participants left. We conducted the interviews in the subsequent weeks. Six months after the experiment, we obtained the points that students achieved in their mandatory *Algorithms and Programming* course.

2.3 Deviations

One of the participants mistakenly joined the *Brocanto* group on the treatment day. That is why the number of participants is not perfectly balanced to 10 students per group; nevertheless, the groups have comparable programming competency according to the pretest. Furthermore, several participants dropped out because the course was too easy for them. This considerably reduces the power of our experiment.

REFERENCES

[1] Alireza Ahadi, Raymond Lister, and Donna Teague. 2014. Falling Behind Early and Staying Behind When Learning to Program. In *PPIG*, Vol. 14.

[2] Lee R Brooks and John R Vokey. 1991. Abstract Analogies and Abstracted Grammars: Comments on Reber (1989) and Mathews et al.(1989). (1991).

[3] Annette Kinder and Anja Assmann. 2000. Learning Artificial Grammars: No Evidence for the Acquisition of Rules. *Memory & Cognition* 28, 8 (2000), 1321–1332.

[4] Barbara J Knowlton and Larry R Squire. 1996. Artificial Grammar Learning Depends on Implicit Acquisition of both Abstract and Exemplar-Specific Information. *Journal of Experimental Psychology: Learning, Memory, and Cognition* 22, 1 (1996), 169.

[5] Bertram Opitz, Nicola K Ferdinand, and Axel Mecklinger. 2011. Timing Matters: the Impact of Immediate and Delayed Feedback on Artificial Language Learning. *Frontiers in human neuroscience* 5 (2011), 8.

[6] Bertram Opitz and Angela D Friederici. 2003. Interactions of the Hippocampal System and the Prefrontal Cortex in Learning Language-Like Rules. *NeuroImage* 19, 4 (2003), 1730–1737.

[7] Bertram Opitz and Angela D Friederici. 2004. Brain Correlates of Language Learning: the Neuronal Dissociation of Rule-Based versus Similarity-Based Learning. *Journal of Neuroscience* 24, 39 (2004), 8436–8440.

[8] AS Reber. 1967. Implicit Learning of Artificial Grammars. J. Verbal Learn. Verbal Behav. 6 (6), 855–863.

[9] Arthur S Reber. 1989. Implicit Learning and Tacit Knowledge. *Journal of experimental psychology: General* 118, 3 (1989), 219.

[10] Janet Siegmund, Christian Kästner, Jörg Liebig, Sven Apel, and Stefan Hanenberg. 2014. Measuring and Modeling Programming Experience. *Empirical Software Engineering* 19, 5 (2014), 1299–1334.

[11] Edward L. Thorndike and R. S. Woodworth. 1901. The Influence of Improvement in One Mental Function upon the Efficiency of Other Functions. *Psychological Review* 8 (1901), 247–261. 3.

[12] Benjamin Xie, Dastyni Loksa, Greg L. Nelson, Matthew J. Davidson, Dongsheng Dong, Harrison Kwik, Alex Hui Tan, Leanne Hwa, Min Li, and Amy J. Ko. 2019. A Theory of Instruction for Introductory Programming Skills. *Computer Science Education* 29, 2-3 (2019), 205–253. https://doi.org/10.1080/08993408.2019.1565235

Table 2: Pretest, exercises per programming construct and the posttest, as well as the posttest programming task and its percentage correctness divided by test and control group. The programming posttest is presented on a scale between 0 and 2.

		Exercise Scores							Posttests Scores		
Participant	Pretest	Data types	Variables	Arithmetic	Print	Logical	Conditionals	Loops	Overall	Programming	AaP
PG1	88	100	88	86	71	95	60	81	100	2	-
PG2	63	100	100	100	100	85	100	89	100	2	42
PG3	0	100	96	81	71	90	68	66	69	0	-
PG4	63	100	92	89	57	90	92	57	88	2	41
PG5	100	100	100	100	100	100	100	15*	100	2	11
Mean	63	100	95	91	80	92	84	73	91	1.6	31.3
PB1	31	100	96	100	100	85	100	81	100	2	-
PB2	38	100	96	94	100	85	96	78	88	2	-
PB3	0	100	100	94	71	80	100	88	100	2	-
PB4	75	100	100	97	100	95	100	98	75	2	-
PB5	94	100	100	100	100	95	96	100	81	2	-
PB6	94	100	96	97	100	95	100	99	100	2	28
PB7	25	100	96	100	71	85	100	63	75	1	-
PB8	100	92	88	-+	-+	100	96	85	100	2	34
Mean	57	99	97	97	92	90	99	87	90	1.9	31

* Did not solve all tasks regarding loops due to illness and is not included in this mean.

+ Did not attend these assignments.

- Cancelled the semester course.