Memory & Cognition
https://doi.org/10.3758/s13421-018-0838-0

Simulation in children's conscious recursive reasoning

M. Bucciarelli[1] · R. Mackiewicz[2] · S. S. Khemlani[3] · P. N. Johnson-Laird[4,5]

Abstract
When do children acquire the ability to understand recursion—that is, repeated loops of actions, as in cookery recipes or computer programs? Hitherto, studies have focused either on unconscious recursions in language and vision or on the difficulty of conscious recursions—even for adults—when learning to program. In contrast, we examined 10- to 11-year-old fifth-graders' ability to deduce the consequences of loops of actions in informal algorithms and to create such algorithms for themselves. In our experiments, the children tackled problems requiring the rearrangement of cars on a toy railway with a single track and a siding—an environment that in principle allows for the execution of any algorithm—that is, it has the power of a universal Turing machine. The children were not allowed to move the cars, so each problem's solution called for them to envision the movements of cars on the track. We describe a theory of recursive thinking, which is based on kinematic simulations and which we have implemented in a computer program embodying *mental models* of the cars and track. Experiment 1 tested children's ability to deduce rearrangements of the cars in a train from descriptions of algorithms containing a single loop of actions. Experiment 2 assessed children's spontaneous creation of similar sorts of algorithms. The results showed that fifth-grade children with no training in computer programming have systematic abilities to deduce from and to create informal recursive algorithms.

Keywords Recursion · Informal algorithms · Deduction · Abduction · Kinematic simulations

In computer science, any process that contains a loop of actions is *recursive* (Enderton, 2010). Recursion is also commonplace in daily life—from cookery recipes to laying place settings on a table. It lies at the core of computation: a loop of actions is repeated either for a given number of times or while a given condition continues to hold—though loops that do not terminate are the bane of programmers. But, what are the origins of recursion? Most people who have thought about this question have assumed that it depends on innate mental machinery (e.g., Hauser, Chomsky, & Fitch, 2002). This assumption is hard to test, but it does raise a more tractable question: When does recursion first appear as children develop?

Recursive rules are part of grammar, and 7-year-old children can already generate recursive sentences (Berwick, Pietroski, Yankama, & Chomsky, 2011; Miller, Kessel, & Flavell, 1970; Roeper, 2009). Likewise, 10-year-olds can discriminate between diagrams that are the products of recursive processes (fractals) and those that are not (Martins, Laaha, Freiberger, Choi, & Fitch, 2014). The application of recursion in these skills is unconscious, but it is exercised in a deliberate and conscious way in writing computer programs. However, not much research has examined whether children can cope with recursion outside calculation or programming. One issue is that the purview of recursive reasoning is often narrow and is thought of as a specialized operation in computer science, in which functions call themselves, or as a sort of reasoning that is self-referential (e.g., Cherubini & Johnson-Laird, 2004). From this perspective, children and adults seldom make recursive inferences. Indeed, this narrow conception of recursion is more relevant to the niceties of logic, computability, and programming, where a function that calls itself is elegant.

Previous studies have examined how children trained in computer programming understand recursion (see Chan

✉ M. Bucciarelli
monica.bucciarelli@unito.it

[1] Dipartimento di Psicologia and Centro di Logica, Linguaggio e Cognizione, Università di Torino, 10123 Torino, Italy

[2] Department of Psychology, University of Social Sciences and Humanities, Warsaw, Poland

[3] Navy Center for Applied Research in Artificial Intelligence, Naval Research Lab, Washington, DC, USA

[4] Stuart Professor of Psychology, Emeritus, Princeton University, Princeton, NJ, USA

[5] Department of Psychology, New York University, New York, NY, USA

Published online: 11 July 2018

Mow, 2008; Mayer, 2013; Sleeman, 1986, for reviews). For example, children's recursive abilities have been examined in programming languages such as LOGO (Papert, 1980) and LEGO (Resnick, 1994), and 10-year-olds have been shown to have difficulty learning the concept of recursion (e.g., Dicheva & Close, 1996), whereas 11-year-olds have difficulty thinking about how recursive programs work (e.g., Kurland & Pea, 1985). However, programming depends on much more than a grasp of recursion: It calls for knowledge of formal programming language.

Any recursive function is equivalent to a loop of operations, and such loops are of two sorts. One sort is specified beforehand to be carried out for a given number of repetitions (a *for* loop), and the other, which can compute functions beyond the scope of *for* loops, is specified to repeat while a particular condition holds (a *while* loop; see, e.g., Enderton, 2010; for an introduction, see Johnson-Laird, 1983, chap. 1). This broader notion of recursion as a loop of operations clarifies why it is commonplace in everyday life—for example, "take two pills a day for five days," "scrub while the stain still shows," or "beat until the cream holds a peak." Recursive reasoning therefore concerns the ability to reason about the repetition of actions. So, in attempting to answer our question about when a conscious grasp of recursion first develops in human life, our main assumption is that this is not a matter of understanding calculation or computer programming. It does not call for specialized training in formal languages and symbols, but instead depends on grasping the broader conception of a repeated loop of actions. We therefore simply need participants who can make kinematic simulations of actions, and fifth-grade children can do so (e.g., Caeyenberghts, Wilson, van Roon, Swinnen, & Smits-Engelsman, 2009; Skoura, Vinter, & Papaxanthis, 2009). We need participants who can plan rearrangements, as in the Tower of Hanoi problem, and fifth-grade children can do this as well (e.g., Aamodt-Leeper, Creswell, McGurk, & Skuse, 2001; Keen, 2011). Finally, we need participants who can solve problems using means–ends analysis, and once again, fifth-grade children can do so (e.g., Kuhn, 2013). We do not claim that younger children cannot cope with recursion in a conscious way, but we do claim that fifth-graders appear to be the best population from which to draw a sample to carry out such recursions at a level better than chance.

The focus of our investigation was on informal algorithms—namely, those described in everyday language. These concerned rearrangements of the order of cars in trains on a toy railway, which we describe in detail below. Readers needed to bear in mind, however, that the railway allows for three different sorts of task to be given to participants:

1. *Problem solving*: The participants have to solve a rearrangement problem by moving the cars from their given order into a required new order, using a siding on the track where necessary.
2. *Deduction*: They have to deduce a new order of cars from a description of an algorithm that makes a rearrangement of a given order.
3. *Abduction*: They have to formulate their own informal algorithm for making a rearrangement. This process of creating an algorithm is a sort of inductive reasoning, but one that is known as "abduction," because it is more akin to an explanation of how to make a rearrangement than to a generalization from the rearrangement.

In an earlier study, we showed that Naïve adults—that is, those who knew nothing about programming or its cognate disciplines—can carry out all three sorts of tasks (Khemlani, Mackiewicz, Bucciarelli, & Johnson-Laird, 2013). The evidence corroborated their use of kinematic mental simulations. Likewise, in a previous study of fifth-grade children, we examined two of the three tasks. We showed that children can solve problems of rearranging five cars, and that they can abduce informal algorithms for rearranging trains of six cars (Bucciarelli, Mackiewicz, Khemlani, & Johnson-Laird, 2016). We also showed that gestures helped them abduce algorithms when they were not allowed to move the cars.

The present investigation was designed to answer two new questions that earlier studies had never addressed: Could fifth-grade children make deductions from algorithms, and could they abduce recursive algorithms for trains of an indefinite length? In Experiment 1, we therefore examined children's ability to deduce the consequences of algorithms presented to them in written form; some of these algorithms were for rearranging trains of five cars, and some of them were recursive, containing a loop of operations appropriate for trains of any length. Experiment 2 examined children's ability to abduce their own informal algorithms for making rearrangements; some of their algorithms had to rearrange trains of six cars, but some of them had to rearrange trains of an indefinite length—that is, correct algorithms had to be recursive and to contain a loop of moves.

In the rest of this introduction, we describe the railway environment and a theory of recursive thinking based on kinematic mental models. Next, we report the two experiments, one on deduction, and one on abduction. We conclude with a general discussion of the implications of their results for alternative theories of recursive reasoning and for the pedagogy of programming.

A domain of recursive problems

Recursion concerns a loop of operations, either a *for* loop or a *while* loop. As far as anyone knows, a system that is equipped with a small number of basic functions can compute anything that can be computed if it can use these functions in recursive loops. The basic functions can be

arithmetical, such as the successor function, which, given a natural number such as 4, returns its successor, 5. But, recursion applies to any sort of operation, so we needed to find a domain suitable for both adults and children who know nothing about programming. The environment that we developed consists of a railway track with cars that can move along the track (see Khemlani et al., 2013). Figure 1 is a diagram of this environment.

As Fig. 1 shows, the track has three parts: the left side, the siding, and the right side. A problem consists of a train on the left side that has to be rearranged into a new order on the right side, such as the rearrangement of ABCDE into EDCBA. This goal has to be achieved by moving cars, one or more at a time, so that they arrive at the required rearrangement on the right track. Three rules govern the movement of cars:

1. Cars can move only along the tracks: One car cannot jump over another. So, when one car moves, it also moves any car in front of it.
2. Only three sorts of move are allowed: Cars can move from the left track to the right track (R), from the left track to the siding (S), and from the siding back to the left track (L). They cannot move from the right track back to the left track, or from the siding straight to the right track.
3. The trains must be rearranged in as few moves as possible, so when it is necessary to move more than one car, the cars should move together.

The siding allows cars to be stored for a while so that other cars can move unimpeded from the left to the right track. The siding is therefore akin to a stack-like memory. But so, too, is the left track, because cars can shuttle between the two in intricate dances, before they move to the right track.

Children and adults have no difficulty understanding the environment and its rules, and in solving problems that call for rearranging cars in a train (Bucciarelli et al., 2016; Khemlani, Goodwin, & Johnson-Laird, 2015; Khemlani et al., 2013). One potential worry is that the environment is idiosyncratic and not representative of recursive domains. However, given the ability to add cars to the track or remove them, and to have the cars denote zeroes and ones, the system is equivalent to a universal Turing machine, because both the left track and the siding are stacks, from a computational standpoint (Hopcroft & Ullman, 1979). In theory, through these simple additions and the possibility of extending the length of each of the three parts of the track as required to accommodate any number of cars, the railway can carry out any computation.

The rearrangement we described earlier is a permutation of the original train, with the order of its cars reversed. Permutations have an interesting property seldom mentioned in texts on the topic (cf. Bona, 2012): A particular permutation, of which there are a countable infinity, can apply to any number of entities. For example, a reversal can apply to trains of any length. Hence, an algorithm for reversals needs to work for any number of cars and is bound to call for at least one recursive loop of operations. That is why we used the railway environment in our studies.

A theory of the abduction of algorithms

An infinite number of algorithms can compute the same function, such as a reversal (Enderton, 2010). The process of formulating an algorithm is therefore akin to the abduction of an explanation: It goes beyond the given information, which needs only to state the inputs and outputs of a function. To abduce an algorithm that solves any instance of a rearrangement, such as a reversal of cars, three steps are necessary.

The first step is to solve one or more instances of the problem. Although there are only three sorts of move, their possible sequences soon overwhelm any attempt at a solution by trial and error. Rearrangements can be solved, however, in a sort of means–ends analysis by working on the rightmost car or cars yet to be solved in the goal rearrangement—that is, a *partial* means–ends procedure. As an illustration, consider a reversal of four cars, from ABCD on the left track to DCBA on the right track. The first goal is to get A over to the right track. It cannot move there, however, without moving the cars in front of it there, too. So, these cars need to be moved out of the way—that is, onto the siding. We represent this move as follows, in which (S 3) denotes a move of three cars to the siding:

ABCD[−]−becomes A[BCD]− (S3)

The square brackets indicate the cars on the siding, of which there are none in the starting situation; any cars to the left of the brackets are on the left track, and any cars to the right of them are on the right track. Now that only car A is on the left

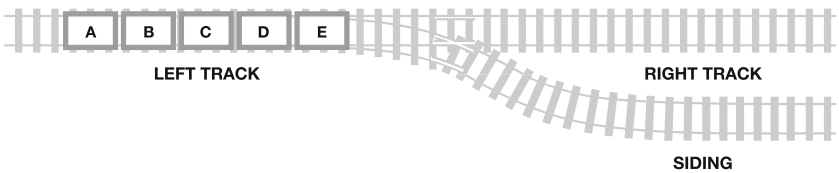

Fig. 1 Diagram of the railway environment for studying recursive thinking: a train of five cars (ABCDE) is on the left track (see the text for the rules governing the movements of cars in the train)

track, it can be moved to the right track:

−[BCD]A (R1)

that is, a move of one car to the right track. Because car A has now been solved, the goal can be updated to DCB. It is easy to solve its rightmost car. We move B from the siding,

B[CD]A (L 1)over to the right track:
−[CD]BA (R 1)

The goal is now DC, and the solution for C calls for the same two moves performed for B in order to get it to the right track:

C[D]BA (L1)
−[D]CBA (R1)

The same two moves are then repeated again to get D into the correct position:

D[−]CBA (L1)
−[−]DCBA (R1)

And the problem is solved.

This partial means–ends analysis can solve any rearrangement, but to guarantee a minimal solution—one with the fewest possible moves—takes some exploration in certain cases. The process could be carried out in actual moves on the railway track, but children do have some ability to simulate moves if they are prohibited from touching the actual cars (Bucciarelli et al., 2016). Instead of performing physical moves, children construct mental models of what cars are where on the railway track. We invite readers to imagine how they would rearrange ABCD so that D is at the back of the train: DBCA. It is not difficult. As you may notice, mental models are iconic, in that their structure corresponds to what they represent (Johnson-Laird, 2006, chap. 2). So, they represent the spatial arrangement of the track in a spatial model and simulate the movements of the cars on the track in a kinematic sequence of models. But mental simulations are also costly: Each move either sets up a new mental model or updates an existing one, so its representation depends on the processing capacity of working memory.

The second step is to use a record of the moves in solutions to a problem to abduce an algorithm for carrying them out. If the algorithm is only to rearrange, say, five or six cars, it could be a mere list of the required sequence of moves, though there is an option for certain problems, such as a reversal, to use a recursive loop. If the algorithm is to rearrange trains with any number of cars, then a loop may be essential. Consider the sequences of moves for a reversal of four cars:

(S 3)(R 1)(L 1)(R 1)(L1 R1)(L1 R1)

and a reversal of five cars:

(S 4)(R 1)(L 1)(R 1)(L1 R1)(L1 R1)(L1 R1)

To abduce the algorithm, one needs to detect a loop in these sequences, as well as any moves that occur before or after it. In one minimal solution of the reversal, there is an initial move of (S *n*–1), where *n* is the number of cars in the train, then a loop of two moves: (R 1)(L 1), and finally a move of (R 1). The general specification of a *for* loop calls for the solution of two simultaneous linear equations, which seems beyond the competence of fifth-grade children. A simpler solution (albeit one that has more computational power) is to simulate the solution and determine the situation that causes a *while* loop to continue. For a reversal, this continues as long as there is at least one car on the siding. Other sorts of problem have different loops with different *while* conditions, but they can be determined from simulations of their solutions.

The third step is to test the algorithm—a step that programmers neglect at their peril—to assess whether it does what it is supposed to do. This step calls for deduction. It simulates the effect of the algorithm on a train of a new length, in order to deduce the consequences of the algorithm and check that the algorithm halts with the required rearrangement on the right track.

A computer program, mAbducer, that the fourth author wrote carries out all three of these steps for any rearrangement that calls for a finite number of moves or a single recursive loop. It is an automatic programmer for rearrangement problems (see Khemlani et al., 2013), and its source code is available at http://mentalmodels.princeton.edu/models/. This automatic programmer generates algorithms, using a *for* loop and a *while* loop, that solve the problems, and it also describes them in both a programming language, Lisp, and informal English. It provided minimal correct algorithms as a basis for the problems in our experiments. The kinematic model that it uses to simulate moves on the track is schematic, and we have already illustrated it in the moves for the reversal above.

In summary, the theory and its computer implementation rest on three assumptions that derive from the theory of mental models—henceforth, the *model theory*, for short. First, simulations depend on *iconic* models. They are iconic in that their structure corresponds to the structure of the world (Johnson-Laird, 1983). Second, they are *kinematic*, in that they unfold in time in the same sequence as the required moves for a problem—that is, they use time to represent time (Hegarty, 2004; Schaeken, Johnson-Laird, & d'Ydewalle, 1996). Third, they are *schematic*, and therefore more parsimonious than visual images, though they may underlie such images. Hence, they yield faster inferences than do images (Knauff, Fangmeier, Ruff, & Johnson-Laird, 2003). A model can therefore represent what is common to many possibilities that differ in their details.

The theory makes three principal predictions. Fifth-grade children should be able to deduce the consequences of algorithms containing loops, and to abduce such algorithms, with better-than-chance accuracy (Prediction 1). They should make more

accurate deductions and abductions for algorithms without loops than for those with loops, because the latter impose an additional load on working memory (Prediction 2). Because simulations depend on the processing capacity of working memory, children should differ in ability (Prediction 3).

Experiment 1: Children's deductions from algorithms

The participants had to deduce the consequences of each of three algorithms for rearranging the order of five cars on a toy railway track. Figure 2 depicts the track at the start of a problem. The algorithms concerned five cars, because this number allowed each algorithm to be presented in two versions: without a loop and with a *while* loop. We used *while* rather than *for* loops so that the children did not have to count the number of iterations of a loop. The three algorithms were for:

1. a reversal of the order of the cars in a train, so the train AEIOU would become UOIEA;
2. a parity sort, in which all the cars in even-numbered positions would be moved in front of all the cars in odd-numbered positions, so that the train AEIOU would become EOAIU;
3. a center palindrome, in which a train would be rearranged by pairing its two outer cars, then the next pair of outermost cars, and so on, until only the center car would be left, which would be put at the end of the train—so the train AEIOU would become AUEOI (see the materials for the algorithms).

Method

Participants The participants were 30 fifth-grade children (16 females and 14 males; mean age 10 years 3 months) attending a primary school in Turin, Italy. The Ethical Committee of the University of Turin approved the experiment, and the children took part in the study after their parents had given their informed consent.

Design The participants deduced the consequences of three sorts of algorithms (reversal, parity sort, and center palindrome) described in one version in a finite list of actions—that is, without a loop and in another version with a *while* loop. The six problems were presented in a different random order to each participant, with the constraint that the two versions of a problem never followed one after the other.

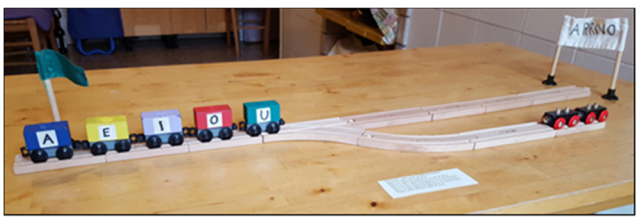

Fig. 2 Railway environment of Experiment 1 at the start of a problem. The banner at the end of the right track says *Arrivo*—that is, "Arrival"

Materials The descriptions of the algorithms were based on mAbducer's outputs, but they were expanded so that each move described both the starting point and the end point of each move—for example, "Move one car from the left track to the right track." As in mAbducer, the algorithms made no reference to the letters that labeled the cars, but concerned only moves of various numbers of cars. The resulting algorithms were translated into Italian, because the children were Italian. Here are the English versions:

1. The reversal algorithm, which reverses the order of the cars:	AEIOU[-]-
Move one less than the number of cars to the siding.	A[EIOU]-
While there are more than zero cars on the siding,	
move one car to the right track,	[EIOU]A
move one car to the left track.	E[IOU]A
Three further iterations of the *while* loop yield:	U[-]OIEA
Move one car to the right track.	-[-]UOIEA

We also used a version of the algorithm in which there was no loop of moves. This version also listed moves to reverse a train of five cars.

2. The parity-sort algorithm puts all the cars in even-numbered positions in front of all the cars in odd-numbered positions:	AEIOU[-]-
While there are more than two cars on the left track,	
move one car to the right track,	AEIO[-]U
move one car to the siding.	AEI[O]U
A further repetition of this loop yields:	A[EO]IU
Move one car to the right track.	[EO]AIU
Move two cars to the left track.	EO[-]AIU
Move two cars to the right track.	-[-]EOAIU

The two final moves were reformulated from mAbducer so that they referred to the specific number of cars required for a train of five cars. We also used a version of the algorithm without a loop. It listed all the required moves for a train of five cars.

3. The center palindrome algorithm transforms a train by pairing the two outer cars, then the next pair of outermost cars, and so on, until only the center car is left and it is put at the end of the train:	AEIOU[-]-
Move two cars from the left track to the siding.	AEI[OU]-
Move one car to the right track.	AE[OU]I
While there are more than zero cars on the left track,	
move one car to the left track,	AEO[U]I
move two cars to the right track.	A[U]EOI
A further repetition of the loop yields the solution:	-[-]AUEOI

To avoid the calculation required in the initial moves, which are irrelevant to the grasp of a loop, we reformulated them as above so that the loop would apply only to trains of five cars.

Procedure The participants were tested one at a time in a quiet room and in the sole presence of the experimenter. They learned the rules for moving cars, and they were told they had to read the description of a series of moves and to work out the effect of these moves on the final order of the cars in the train on the right side of the track. They read the description of the algorithm, which remained in view throughout the complete trial. We video-recorded the experimental sessions, and later transcribed them.

Results and discussion

The data from five of the 30 children were excluded from the analysis because either the children moved cars when solving a problem or a technical error occurred. The statistical analyses were performed on the remaining 25 participants. The analyses assessed whether the group of children as a whole was able to solve the problems at a level better than chance, whether they were more accurate with the problems without loops than with the problems with loops, and whether they differed in ability.

A train of five cars has 5! (= 120) possible rearrangements. Hence, the probability of solving a problem by chance would be 1 in 120, and any problem for which two or more children deduced the solution would be solved more often than chance ($p < .02$). Table 1 presents the numbers of children who made correct deductions for each of the six sorts of algorithm. As these numbers show, the children performed better than chance for each of the six problems in deducing the consequences of informal algorithms. The chance probability of a child solving one or more of the six problems was equal to $1 - (119/120)^6 = .049$. Hence, the group of children as a whole performed better than chance (Prediction 1), because 22 children out of the 25 solved at least one problem (binomial test, $p < .0001$). So, children from the population we sampled should cope with at least one of the problems.

The children were more accurate in deducing the consequences of algorithms without loops (53% correct) than of algorithms with loops (35% correct; Wilcoxon test, $z = 2.12$, $p < .02$, Cliff's $\delta = .30$; Prediction 2). An analysis of the individual problems showed that only the reversal yielded a reliable difference in difficulty: It was easier in the algorithm without a loop than in the algorithm with a loop (Wilcoxon test: $z = 2.33$, $p < .02$, Cliff's $\delta = .28$). The six problems differed in difficulty [Cochran's Q test: $\chi^2(5) = 28.31$, $p < .001$]. It may be that reversals are easy because they repeat a loop of two moves of single cars three times, so the children can grasp the loop better than they can the loops in the other algorithms, which repeat only twice. But any definitive explanation would call for a much larger sample of different rearrangements, of which, in principle, a countable infinity exist.

The children themselves differed in their ability to make accurate deductions from algorithms [Friedman nonparametric analysis of variance: $\chi^2(5) = 28.31$, $p < .0001$: Prediction 3]. Three children made no correct deductions, and two children made only correct deductions. There was no reliable difference in accuracy between the sexes: Boys were 50% correct, and girls were 38% correct (Mann–Whitney test: $z = .99$, $p = .32$, Cliff's $\delta = .23$).

Experiment 1 corroborated the three predictions of the model theory. The children as a group deduced the consequences of each sort of algorithm much better than chance, they were more accurate for algorithms without loops than for algorithms with loops, and they differed in ability. We therefore devised Experiment 2 to find out whether children from the same population could themselves abduce algorithms containing loops of operations.

Table 1 Numbers of children ($N = 25$) in Experiment 1 who made correct deductions of the rearrangements of cars according to three sorts of algorithms, either without loops or with loops

Sort of Algorithm	Without Loop	With Loop
Reversal	20	13
Parity sort	11	5
Center palindrome	9	8

Experiment 2: Children's abduction of algorithms

Our previous study of children had shown that they could abduce algorithms for rearranging trains of six cars (Bucciarelli et al., 2016). An open question was whether they could abduce recursive algorithms to rearrange trains of an indefinite length. Experiment 1 in the present study implied that they could make deductions from such algorithms. Hence, Experiment 2 compared their ability to abduce algorithms for rearranging trains of six cars and trains of an indefinite number of cars, which necessarily required a loop of moves. Each child tackled five pairs of problems: In each pair, the first problem had six cars, and the second problem had an indefinite number of cars. Figure 3 shows the initial state of such a problem, with a tunnel hiding an indefinite number of cars on the left track. The children were told it hid an unknown number of cars at either the front or the back of the train (depending on the location of the tunnel), but leaving six cars visible. The model theory predicts that fifth-graders should be able to abduce algorithms better than chance for both sorts of problem (Prediction 1), that they should be more accurate for trains of

Fig. 3 Initial state of a problem in Experiment 2 that depicts an indefinite number of cars (see the tunnel on the left side of the track and the picture of the goal rearrangement behind the right track)

six cars than for trains of an indefinite length (Prediction 2), and that they should differ in ability (Prediction 3).

Method

Participants The participants were 35 fifth-grade children (16 females and 19 males; mean age 11 years) attending three primary schools in Turin, Italy. The Ethics Committee of the University of Turin approved the study, and the children took part after their parents had given informed consent.

Design The experiment examined the children's ability to formulate five different sorts of algorithm, both for six cars and for an indefinite number of cars, with six cars visible and an unknown number of cars hidden in a tunnel. Each pair of problems had a six-car problem first and then the corresponding problem with an indefinite number of cars. We used this procedure to prevent the children from being dismayed by encountering a difficult problem without any preparation for it. Table 2 summarizes the initial and final states of each problem. The Appendix shows the five algorithms with loops that the children needed to abduce for problems with an indefinite number of cars. The order of the five pairs of problems was random for each child.

Materials and procedure The five sorts of problem, each in the two versions, are illustrated in Table 2. We used white cars labeled with letters (A, B, C, D, E, F), a cardboard tunnel, and photographs of the required rearrangements of the cars. The participants were tested one by one in a quiet room with only the experimenter present. They carried out an initial training in which they learned the rules for moving the cars and how to describe the moves using only the number of cars in a move, without referring to the cars by letter. They were told that the cars on the left track had to be rearranged into the order shown in the photograph behind the right track. They were also told that some trains had an unknown number of cars, so they would have to describe a method of rearranging a train of any length—that is, the tunnel hid many cars, and "we do not know how many."

The key instructions began with these sentences for six cars: "Try to tell me in words, without moving the cars, how would you form this train [in the picture]. Remember not to use the names of the cars, but tell me how many cars move from one track or another." Once the child had created an algorithm, the experimenter introduced the tunnel and reminded the child that it hid an unknown number of cars, which were part of the train that the child could see. The experimenter then constructed these trains of indefinite length in front of the child, who understood that the tunnel hid an unknown number of cars. The instructions for these recursive problems were: "Now, because we do not know how many cars there are in this train, we need rules that summarize the moves to form the train in the picture. The rules must be as short as possible: you must use the smallest number of words." We video-recorded the experimental sessions and later transcribed the children's algorithms.

Results and discussion

Coding of algorithms and of loops Two independent judges coded the video-recordings to make explicit each algorithm

Table 2 Initial and final states of the ten problems in Experiment 2

Names of the Problems	Initial States of the Two Versions	Final States of the Two Versions
1. Swap adjacent pairs	FEDCBA ■FEDCBA	EFCDAB ■EFCDAB
2. Reversals	FEDCBA ■FEDCBA	ABCDEF ABCDEF■
3. Parity sort	FEDCBA ■FEDCBA	FDBECA ■FDB■ECA
4. Back-to palindrome	AABBCC AABBCC■	ABCCBA ABC■CBA
5. Two-loop palindrome	CCBBAA ■CCBBAA	ABCCBA ABC■CBA

The ■ symbol denotes the tunnel, which hides an indefinite number of cars in the initial state of a problem and denotes an indefinite number of cars in the final state of a problem

from the children's verbal descriptions and gestures. The judges also noted the occurrence of any loops, distinguishing three sorts:

- *While* loops specify the termination condition in advance—for example, "and so on until the cars are finished."
- *For* loops specify the number of iterations in advance, though they might do so using a quantifier such as "all," to refer to the unknown number of cars in a train—for example: ". . . we do like that for all the cars we can't see," "one by one take the cars and lead them back [to the left track] and then to the goal."
- *Proto*-loops specify neither the termination condition nor the number of iterations, but indicate that the same move will be repeated—for example, "and so on," "and we go always like that," and "we move the car from the side to the left then to the goal, and also the last one."

The two independent judges agreed in their coding of the algorithms on 92% of trials (Cohen's κ = .84, p < .0001). They also agreed on 97% of trials about the occurrence of no loops, proto-loops, *for* loops, and *while* loops in the algorithms (Cohen's κ = .94, p < .0001). They resolved the discrepancies in both codings prior to the statistical analyses. Because the children often used quantifiers, such as "all the cars," the *while* and *for* loops differed less in their informal versions than they do in formal programs, because the children described *for* loops without explicit numbers of required repetitions.

Table 3 presents typical protocols from two children abducing algorithms for swapping adjacent pairs of cars in trains of an indefinite length. The table also presents the transcription of the protocols into the notation used by the mAbducer program. The cars hidden in the tunnel are denoted by the ■ symbol. The two protocols illustrate a *while* loop and then a *for* loop.

Table 3 Translation from Italian of two children's algorithms in Experiment 2 for swapping adjacent pairs of cars in a train of indefinite length, and their transcriptions into mAbducer's notation

Move	Descriptions and Gestures	Transcription of the Move
Participant 8's transcript for an algorithm with a *while* loop that swaps adjacent pairs to rearrange ■FEDCBA into ■EFCDAB		
1	"One to the siding . . ." (*draws in the air a trajectory from the left track to the siding*)	■ FEDCB[A]
2	". . . the other to the goal." (*draws in the air a trajectory from the left track to the right track*)	■ FEDC[A]B
3	"One on the siding goes back then to the goal . . ." (*draws in the air a trajectory from the siding to the left track and then to the right track*)	■ FEDCA[-]B ■ FEDC[-]AB
4	". . . and so on until all the cars are finished." (*moves one hand in front of the other in a continuous movement in a wheel-like movement*) The description is of a *while* loop, because it indicates moves applied to many cars and states the termination condition.	■ FED[C]AB ■ FE[C]DAB ■ FEC[-]DAB ■ FE[-]CDAB ■ F[E]CDAB ■ [E]FCDAB ■ E[-]FCDAB ■ [-]EFCDAB
Participant 10's transcript for an algorithm with a *for* loop that swaps adjacent pairs		
1	"One should always put a car to the siding . . ." (*P10 made no gestures*)	■ FEDCB[A]
2	". . . and one to the goal . . ."	■ FEDC[A]B
3	". . . then the one on the siding goes back . . . to the goal."	■ FEDCA[-]B ■ FEDC[-]AB
4	"One to the siding . . ."	■ FED[C]AB
5	". . . and the other to the goal."	■ FE[C]DAB
6	"One back and then to the goal . . ."	■ FEC[-]DAB ■ FE[-]CDAB
7	". . . and swap them, and do that for all the (cars of the) train." The assertion is a *for* loop, because it indicates moves applied to many cars, specifying all of them in advance.	■ F[E]CDAB ■ [E]FCDAB ■ E[-]FCDAB ■ [-]EFCDAB

The brackets denote the contents of the siding; any cars to the left of the brackets are on the left track, and any cars to the right of the brackets are on the right track. The ■ symbol stands for the cars hidden in the tunnel

Statistical analysis The statistical analyses were designed to assess whether the group of children as a whole were able to formulate algorithms at a level better than chance, whether they were more accurate in their algorithms for trains of six cars than for trains of indefinite length, and whether they differed in ability.

A train of six cars has 6! (= 720) possible rearrangements, so any problem that at least two of the 35 children solved was one for which solutions occurred much more often than chance ($p <$ .002). Table 4 states the numbers of correct algorithms for the five pairs of problems. These numbers show that children performed much better than chance with all the six-car problems, and better than chance with all but one of the indefinite problems. The chance probability of a child solving one or more of the ten problems was equal to $1 - (719/720)^{10} = .014$. Hence, the group of children as a whole performed better than chance, because 34 children out of the 35 solved at least one problem (binomial test, $p < .0001$). Children in a sample of fifth-graders were therefore able to formulate at least one algorithm (Prediction 1). They were also able to formulate a recursive algorithm better than chance: 22 out of the 35 children did so for at least one problem (binomial test, $p < .0001$).

The children were more accurate in abducing algorithms for trains of six cars (66% correct) than for trains of indefinite length (15% correct; Wilcoxon test: $z = 5.16$, $p < .0001$, Cliff's $\delta = .88$; Prediction 2). The same result occurred for each of the five pairs of problems (in Wilcoxon tests, z ranged from 2.8 to 4.7, p ranged from < .005 to < .0001, and Cliff's δ ranged from .23 to .63). The children used loops in 10% of their algorithms for trains of six cars, and in 67% of their algorithms for trains of indefinite length, whether the algorithms were right or wrong (Wilcoxon test: $z = 4.02$, $p < .0001$, Cliff's $\delta = .51$). The ten problems differed in difficulty [Cochran's Q test: $\chi^2(9) = 133.36$, $p < .001$]. The algorithm for swapping adjacent pairs was easy, for both six cars and indefinite numbers of cars, perhaps because it is a single loop of one-car moves that is repeated three times for a six-car problem. Likewise, the loop for reversals, as we mentioned before, is also simple. In contrast, the palindrome is the most difficult, if only because its algorithm uses two separate loops. The Appendix has descriptions of all five recursive algorithms. It also shows that their Kolmogorov complexity—the number of symbols required to describe them in the formal language of the mAbducer notation—predicts their rank order of difficulties for the children (see Khemlani et al., 2013, for the similar success of this metric for adult participants).

Table 4 Numbers of children ($N = 35$) in Experiment 2 who made correct abductions of five sorts of algorithm for trains of six cars and for trains of indefinite length

Sorts of Algorithm	Length of Trains	
	Trains of Six Cars	Trains of Indefinite Length
Swap adjacent pairs	27	19
Reversal	25	4
Parity sort	22	2
Back-to palindrome	22	0
Two-loop palindrome	19	1

The children differed in ability to abduce the algorithms [Friedman nonparametric analysis of variance: $\chi^2(9) = 133.36$, $p < .0001$; Prediction 3]. One child abduced no correct algorithms, whereas the most accurate children abduced five correct algorithms. The difference in accuracy between the sexes was not reliable: Boys were 37% correct, girls were 44% correct (Mann–Whitney test: $z = 1.03$, $p = .30$, Cliff's $\delta = .20$).

The most striking result was that 22 out of the 35 children formulated at least one correct recursive algorithm, which contained a loop of operations. This result shows that a sample of fifth-grade children performed reliably better than chance at the task. To the best of our knowledge, no previous study has obtained such a result. The results also corroborated our earlier finding that children could abduce algorithms for trains of six cars (Bucciarelli et al., 2016). In sum, the results corroborated the three predictions of the model theory. The children as a group deduced the consequences of each sort of algorithm rather better than chance, they were more accurate for algorithms without loops than for algorithms with loops, and they differed in ability.

General discussion

Fifth-grade children, 10 to 11 years old, have some ability to cope with recursive loops in informal algorithms. The children as a whole in Experiment 1 were able to make deductions from algorithms at a level better than chance (Prediction 1). More than half of the children could deduce the consequences of a recursive algorithm for reversing the order of the cars in a train on a track:

Move one less than the number of cars to the siding.
While there is at least one car on the siding,
move one car from the left track to the right track,
move one car from the siding to the left track.
Move one car to the right track.

Even though they were not allowed to move the actual cars on the track, they could imagine the effects of this recursive algorithm. It was easier for them to deduce the consequences of algorithms that were lists of actions rather than those that were recursive, containing a loop of moves (Prediction 2), as in the preceding example. The three algorithms differed in difficulty, and the one above was the easiest, perhaps because it has a loop of two moves of single cars that is repeated more often than are the loops in the other two algorithms. The repetition of simple moves could

help children deduce the moves' consequences, but other factors may be in play, such as the load on working memory. The space of possible rearrangements is boundless, and without results from a larger sample of algorithms, it is impossible to draw definite conclusions. Congruent with the role of working memory, however, the children in Experiment 1 differed in their ability to deduce the consequences of algorithms (Prediction 3).

Fifth-grade children can also abduce their own informal algorithms containing loops of moves. The sample as a whole in Experiment 2 were able to do so at a level better than chance (Prediction 1). This result contrasts with earlier findings that in computer programming, fifth-graders have difficulty coping with recursion (e.g., Dicheva & Close, 1996; Kurland & Pea, 1985). Yet, like deduction, the abductive task was easier for them when a list of actions sufficed for trains of six cars than when it called for a loop of actions on trains of indefinite length (Prediction 2). The five algorithms differed in difficulty; the Appendix presents them and shows that their Kolmogorov complexity predicts their difficulty (as it had for the adult participants in Khemlani et al., 2013). Again, the difference in the children's abilities suggests that the load on working memory affects performance.

What does it take for you to abduce a recursive algorithm? Our studies corroborated the model theory described earlier in the article. It postulates that you need three interrelated abilities. First, you have to be able to solve the problems that the algorithm is going to solve. In the railway environment, you can do so using the partial means-ends analysis. You can work backward from each car at the head of the goal. If you cannot carry out the actual moves on the track—and the participants in the Bucciarelli et al. (2016) study were not allowed to—then you have to imagine them. So, you carry out a *kinematic simulation* of the solution. But, solutions of rearrangements are not enough for an abduction.

Second, you have to remember the sequence of moves in your simulation, and to abduce its structure, discovering any loop of moves, and any moves before or after the loop. For example, in reversing four cars you made the following sequences of moves:

(S 3)(R 1)(L 1)(R 1)(L 1)(R 1)(L 1)(R 1)

You may then be able to abduce a general procedure. Given that n is the number of cars in the train, you start with a move of $n - 1$ cars to the siding. Next, you make the move (R 1). Then, there is a loop of (L 1)(R 1), which continues while there is at least one car on the siding. So, one minimal algorithm is:

(S (n – 1))
(R 1)
((L 1)(R 1)) while there is at least one car on the siding.

There is an alternative minimal algorithm, which we described earlier (see also the Appendix).

The third and final step is to test your algorithm. You deduce its consequences for a train of a new length. You carry out a mental simulation of it on a longer train. You apply each of its moves on the train, and you check that the end result matches the required reversal. If it does, then the algorithm is complete, assuming that you can describe it (cf. Table 3).

Does any alternative theory of the abduction of algorithms give a different account of representation and process? Cognitive scientists have pursued many accounts of mental representations. Some have claimed that they are not required for intelligent behavior (e.g., Brooks, 1991). Others have argued against a causal role for visual images, and posited instead "mentalese"—that is, a language of thought made up of strings of symbols (e.g., Pylyshyn, 2003). In fact, it was impossible to formulate algorithms in our study without using mental representations. One that may be optimal for formulating an algorithm is a kinematic model. It is iconic, in that it uses time to represent time and that its spatial relations represent spatial relations in the world (see also Hegarty, 2004). Such mental models can underlie visual images, or they may be as abstract as the notation in the present article—a notation that the mAbducer program uses. Indeed, a brain-imaging study has shown that people can reason from models without transforming them into visual images, which in fact impede reasoning (Knauff et al., 2003). Not all reasoning has to depend on iconic models: People who are taught logic can also learn to use formal rules of inference. Likewise, the model theory relies on a representation of meaning that is not iconic, and it is from this representation that it constructs models (Khemlani & Johnson-Laird, 2013). Theorists could argue that all representations rest on such a mentalese, which in turn rests on nerve impulses—just as mAbducer's representations rest on machine language, which in turn rests on an electronic binary code. In both cases, however, the abduction of algorithms demands a higher level of representation, one that humans can envisage and manipulate consciously.

The abduction of loops takes the results of simulations as its input. Probabilities play no essential role, children's protocols make no reference to them, and so Bayesian inferences seem irrelevant (pace Oaksford & Chater, 2007). But, the process could in principle be carried out in first-order logic (see Rips, 1994). It would depend on an axiomatization of rearrangements in the railway domain. The set of axioms, however, would have to capture both what changes and what does not change with each move, that is, they would need to solve the so-called "frame" problem (see Shanahan, 2016). Moreover, logical expressions are not iconic. For example, the spatial relations in this premise are not represented in an iconic way:

$\exists!x\exists!y((\text{car } x) \wedge (\text{train } y) \wedge (\text{at-front-of } x\ y) \wedge (\text{on left-track } y)$

(There is a car, x, and a train, y, such that x is at the front of y and y is on the left track.)

Hence, logical systems for spatial reasoning tend to make the wrong predictions of difficulty (see, e.g., Jahn, Knauff, & Johnson-Laird, 2007; Schaeken, Girotto, & Johnson-Laird, 1998). In contrast, iconic diagrams improve both the accuracy and speed of reasoning in comparison with noniconic verbal premises (Bauer & Johnson-Laird, 1993). It therefore seems that mental simulations should be based on iconic models rather than on logical expressions. The crux of abduction is to discover repeated sequences of operations. The mAbducer program finds them using a recursive process that starts with loops of half the length of the sequence of moves in a solution, and works its way downward through shorter lengths. Human reasoners must also search for loops, but they are fallible, and some patterns are too difficult for them to detect (Khemlani et al., 2013).

Mathematicians, programmers, and cognitive scientists reason about recursion. And many psychological studies have investigated novice programmers trying to formulate algorithms in a programming language (e.g., Kurland & Pea, 1985). Other studies have used arithmetic. For example, teenagers are better at calculating an arithmetical function when it is expressed in an iterative loop of operations than in a function that calls itself (Anzai & Uesato, 1982). However, those who start with iterative calculations do better than those who start with the self-referential calculations. Likewise, the experience of informal algorithms in the railway setting could help budding programmers to master recursive functions. It might even provide a transparent environment in which to teach programming. No valid test exists for predicting the programming ability of individuals who know nothing about it. Children differed in their skill in abducing algorithms for rearrangements, and so the task could predict their ability to program.

Recursion underlies languages (e.g., Hauser et al., 2002), counting and arithmetic (e.g., Enderton, 2010), theory of mind (e.g., Corballis, 2011), and the recognition of visual patterns (e.g., Martins, Mursic, Oh, & Fitch, 2015). One controversy concerns whether all these cases of recursion are rooted in language (Hauser et al., 2002) or else in several cognitive domains (Jackendoff & Pinker, 2005). In a study of an agrammatical patient, Zimmerer and Varley (2010) showed that recursion was absent in the patient's grammar, but present in other domains, such as arithmetic. As far as we know, no-one has established a double dissociation between language and recursion. But, in many recursive domains, such as the formulation of grammatical sentences or inferences from the theory of mind, the underlying recursive principles are unconscious. In the present studies, however, children were conscious of explicit loops of actions from which they had to make deductions, and they attempted to create such loops in abducing informal programs. Both tasks can be carried out using the symbols of an artificial language, such as the notation for mAbducer or LOGO, and without any overt use of natural language. They could therefore provide a test bed to examine the recursive abilities of individuals bereft of language and even of members of other species.

In conclusion, our study of fifth-graders' grasp of recursion revealed that they can deduce the consequences of some algorithms, and that they can abduce the loops of moves required for some algorithms. They are more accurate with algorithms that are lists than with algorithms that include loops. Simulations appear to be crucial: They unfold in time in a sequence of kinematic models, which have to be held in working memory, and processing capacity is therefore critical. This may account for the differences in ability from one child to another. Recursion is an unconscious foundation for many human skills, from perception to speech. It is a conscious component of logic and programming. The informal mastery of our participants in tasks for which they had no explicit preparation suggests that its roots are part of human competence. This ability may be founded on the simulation of sequences of events, and in turn on the ability to make such simulations the objects of conscious thought.

Author note The data for this article are archived in a database to be found at https://osf.io/jg2fy. The research was funded in part by the Polish National Science Centre [Grant 2014/14/*M*/HS6/00916] (to R.M.). We are grateful to Matthew Traxler and the anonymous reviewers for their helpful advice and criticisms of earlier drafts.

Appendix: The algorithms to be abduced in Experiment 2

Five algorithms had to be abduced for trains of both six cars and an indefinite number of cars. An infinite number of algorithms can compute a given rearrangement, and even more than one minimal algorithm exist for some rearrangements. For example, there are two minimal algorithms for a reversal (see below). We summarize the five simplest algorithms with *while* loops that cope with an indefinite number of cars. The notation used here can be illustrated as follows:

(R 1): denotes a move of one car from the left track to the right track,
(S ½*n*): denotes a move of half the number of cars, *n*, in the train from the left track to the siding,
(L 2): denotes a move of two cars from the siding to the left track.
((L 1)(R 1)), while there is at least one car on the siding: denotes a *while* loop.

The five recursive algorithms that the children had to abduce were as follows.

1. An algorithm to swap adjacent pairs, e.g., FEDCBA becomes EFCDAB:

   ((S 1)(R 1)(L 1)(R 1)) while there is at least one car is on left-track

2. An algorithm for reversals, e.g., FEDCBA becomes ABCDEF:

   (S (n – 1))
   (R 1)
   ((L 1)(R 1)) while there is at least one car on the siding

   An alternative minimal algorithm is:

   (S (n – 1))
   ((R 1)(L 1)) while there is at least one car on the siding
   (R 1)

3. An algorithm for parity sorts, e.g., FEDCBA becomes FDBECA:

   (R 1)
   ((S 1)(R 1)) while there is more than one car on the left track
   (L ½n – 1)
   (R ½n)

4. An algorithm to get back to a palindrome, e.g., AABBCC becomes ABCCBA:

   (S (n – 2))
   ((R 1)(L 2)) while there is at least one car on the siding
   (R (½n + 2))

5. An algorithm for a two-loop palindrome, e.g., CCBBAA becomes ABCCBA:

   ((S 1)(R 1)) while there is more than one car on the left-track
   ((L1)(R 1)) while there is at least one car on the siding

Why do these five algorithms differ in the ease with which the participants can abduce them? Several factors seem pertinent. Swapping adjacent pairs should be easy, because it is a single loop in which only the two cars at the current head of the train on the left track are moved: The first car is moved out of the way (to the siding) so the second car can move to the right, and then the first car moves off the siding and over to the right. This sequence is repeated three times for six cars. The two-loop palindrome should be very difficult, because it is the only algorithm that calls for two separate loops of moves. In a study with adults and a different set of algorithms (Khemlani et al., 2013), Kolmogorov complexity predicted the difficulty of abducing recursive algorithms. This complexity depends on the number of characters needed to describe an algorithm in standard language (Li & Vitányi, 1997). The numbers of symbols, including words and parentheses, in the descriptions above of the five algorithms are, respectively, 20, 32, 36, 37, and 40, and the statements of the *while* conditions require ten words for each of the algorithms. These numbers for K-complexity correlate with the numbers of participants who failed to abduce the algorithms: 16, 31, 33, 35, 34 (cf. Table 4; Kendall's $\tau = .84$, $z = 2.06$, $p < .02$).

All the loops in these problems are static, but other problems, which we avoided in the present study, are dynamic, in that the number of cars for at least one move in the loop depends on both the length of the train and the number of repetitions of the loop that have occurred. For example, the "riffle in" shuffle (a.k.a. the Faro shuffle) interpolates the cars in even-numbered positions between those in odd-numbered positions. For example: ABCDEF becomes ADB ECF, and, as mAbducer discovered, it calls for a dynamic algorithm in which the number of cars included in moves in the loop decreases on each repetition. Such an algorithm may be beyond the competence of even naive adults to abduce.

References

Aamodt-Leeper, G., Creswell, C., McGurk, R., & Skuse, D. H. (2001). Individual differences in cognitive planning on the Tower of Hanoi task: Neuropsychological maturity or measurement error? *Journal of Child Psychology and Psychiatry*, *42*, 551–556.

Anzai, Y., & Uesato, Y. (1982). Learning recursive procedures by middleschool children. In Proceedings of the Fourth Annual Conference of the Cognitive Science Society (pp. 100–102). Hillsdale: Erlbaum.

Bauer, M. I., & Johnson-Laird, P. N. (1993). How diagrams can improve reasoning. *Psychological Science*, *4*, 372–378. https://doi.org/10.1111/j.1467-9280.1993.tb00584.x

Berwick, R. C., Pietroski, P., Yankama, B., & Chomsky, N. (2011). Poverty of the stimulus revisited. *Cognitive Science*, *35*, 1207–1242.

Bona, M. (2012). Combinatorics of permutations (2nd). Boca Raton: Taylor & Francis.

Brooks, R. (1991). Intelligence without representation. *Artificial Intelligence*, *47*, 139–160.

Bucciarelli, M., Mackiewicz, R., Khemlani, S. S., & Johnson-Laird, P. N. (2016). Children's creation of algorithms: Simulations and gestures. *Journal of Cognitive Psychology*, *28*, 297–318.

Caeyenberghts, K., Wilson, P. H., van Roon, D., Swinnen, S. P., & Smits-Engelsman, B. C. M. (2009). Increasing convergence between imagined and executed movement across development: Evidence for the emergence of movement representations. *Developmental Science*, *12*, 474–483.

Chan Mow, I. (2008). Issues and difficulties in teaching novice computer programming. In M. Iskander (Ed.), Innovative techniques in

instruction technology, e-learning, e-assessment (pp. 199–204). New York: Springer.
Cherubini, P., & Johnson-Laird, P. N. (2004). Does everyone love everyone? The psychology of iterative reasoning *Thinking & Reasoning*, *10*, 31–53.
Corballis, M. C. (2011). The recursive mind: The origins of human language, thought, and civilization. Princeton: Princeton University Press.
Dicheva, D., & Close, J. (1996). Mental models of recursion. *Journal of Educational Computing Research*, *14*, 1–23.
Enderton, H. B. (2010). Computability theory: An introduction to recursion theory. San Diego: Academic Press.
Hauser, M. D., Chomsky, N., & Fitch, W. T. (2002). The faculty of language: What is it, who has it, and how did it evolve? *Science*, *298*, 1569–1579. https://doi.org/10.1126/science.298.5598.1569
Hegarty, M. (2004). Mechanical reasoning by mental simulation. *Trends in Cognitive Sciences*, *8*, 280–285.
Hopcroft, J. E., & Ullman, J. S. (1979). Introduction to automata theory, languages, and computation (1st). Boston: Addison-Wesley.
Jackendoff, R., & Pinker, S. (2005). The nature of the language faculty and its implications for evolution of language (Reply to Fitch, Hauser, and Chomsky). *Cognition*, *97*, 211–225.
Jahn, G., Knauff, M., & Johnson-Laird, P. N. (2007). Preferred mental models in reasoning about spatial relations. *Memory & Cognition*, *35*, 2075–2086.
Johnson-Laird, P. N. (1983). Mental models: Towards a cognitive science of language, inference, and consciousness. Cambridge: Harvard University Press.
Johnson-Laird, P. N. (2006). How we reason. Oxford: Oxford University Press.
Keen, R. (2011). The development of problem solving in young children: A critical cognitive skill. *Annual Review of Psychology*, *62*, 1–21.
Khemlani, S., Goodwin, G. P., & Johnson-Laird, P. N. (2015). Causal relations from kinematic simulations. In D. C. Noelle, R. Dale, A. S. Warlaumont, J. Yoshimi, T. Matlock, C. D. Jennings, & P. P. Maglio (Eds.), Proceedings of the 37th Annual Conference of the Cognitive Science Society (pp. 1075–1080). Austin: Cognitive Science Society.
Khemlani, S. S., & Johnson-Laird, P. N. (2013). The processes of inference. *Argument & Computation*, *4*, 4–20.
Khemlani, S. S., Mackiewicz, R., Bucciarelli, M., & Johnson-Laird, P. N. (2013). Kinematic mental simulations in abduction and deduction Proceedings of the National Academy of Sciences, *110*, 16766–16771.
Knauff, M., Fangmeier, T., Ruff, C. C., & Johnson-Laird, P. N. (2003). Reasoning, models, and images: Behavioral measures and cortical activity. *Journal of Cognitive Neuroscience*, *15*, 559–573.
Kuhn, D. (2013). Reasoning. In P. D. Zelazo (Ed.), The Oxford handbook of developmental psychology (pp. 744–764). Oxford: Oxford University Press.
Kurland, D. M., & Pea, R. D. (1985). Children's mental models of recursive Logo programs. *Journal of Educational Computing Research*, *1*, 235–243.
Li, M., & Vitányi, P. (1997). An introduction to Kolmogorov complexity and its applications (2nd). New York: Springer.
Martins, M., Mursic, Z., Oh, J., & Fitch, W. T. (2015). Representing visual recursion does not require verbal or motor resources. *Cognitive Psychology*, *77*, 20–41.
Martins, M. D., Laaha, S., Freiberger, E. M., Choi, S., & Fitch, W. T. (2014). How children perceive fractals: Hierarchical self-similarity and cognitive development. *Cognition*, *133*, 10–24.
Mayer, R. E. (2013). Teaching and learning computer programming: Multiple research perspectives. London: Routledge.
Miller, P. H., Kessel, F. S., & Flavell, J. H. (1970). Thinking about people thinking about people thinking about . . . : A study of social cognitive development. *Child Development*, *41*, 613–623.
Oaksford, M., & Chater, N. (2007). Bayesian rationality: The probabilistic approach to human reasoning. New York: Oxford University Press.
Papert, S. (1980). Mindstorms. New York: Basic Books.
Pylyshyn, Z. (2003). Return of the mental image: Are there really pictures in the brain? *Trends in Cognitive Sciences*, *7*, 113–118. https://doi.org/10.1016/S1364-6613(03)00003-2
Resnick, M. (1994). Turtles, termites, and traffic jams: Explorations in massively parallel microworlds. Cambridge: MIT Press.
Rips, L. J. (1994). The psychology of proof. Cambridge: MIT Press.
Roeper, T. (2009). The minimalist microscope: How and where interface principles guide acquisition. In J. Chandlee, M. Franchini, S. Lord, & G. M. Rheiner (Eds.), Proceedings of the 33rd Annual Boston University Conference on Language Development (pp. 24–48). Medford: Cascadilla Press.
Schaeken, W. S., Girotto, V., & Johnson-Laird, P. N. (1998). The effect of an irrelevant premise on temporal and spatial reasoning. *Kognitionswisschenschaft*, *7*, 27–32.
Schaeken, W. S., Johnson-Laird, P. N., & d'Ydewalle, G. (1996). Mental models and temporal reasoning. *Cognition*, *60*, 205–234.
Shanahan, M. (2016). The frame problem. In E. N. Zalta (Ed.), *The Stanford encyclopedia of philosophy*. https://plato.stanford.edu/entries/frame-problem/
Skoura, X., Vinter, A., & Papaxanthis, C. (2009). Mentally simulated motor actions in children. *Developmental Neuropsychology*, *34*, 356–367.
Sleeman, D. (1986). The challenges of teaching computer programming. *Communications of the ACM*, *29*, 840–841.
Zimmerer, V., & Varley, R. A. (2010). Recursion in severe agrammatism. In H. van der Hulst (Ed.), Recursion and human language (pp. 393–405). Berlin: De Gruyter.