ARTIFICIAL INTELLIGENCE

Beyond imitation: Zero-shot task transfer on robots by learning concepts as cognitive programs

Miguel Lázaro-Gredilla*, Dianhuan Lin, J. Swaroop Guntupalli, Dileep George*

Humans can infer concepts from image pairs and apply those in the physical world in a completely different setting, enabling tasks like IKEA assembly from diagrams. If robots could represent and infer high-level concepts, then it would notably improve their ability to understand our intent and to transfer tasks between different environments. To that end, we introduce a computational framework that replicates aspects of human concept learning. Concepts are represented as programs on a computer architecture consisting of a visual perception system, working memory, and action controller. The instruction set of this cognitive computer has commands for parsing a visual scene, directing gaze and attention, imagining new objects, manipulating the contents of a visual working memory, and controlling arm movement. Inferring a concept corresponds to inducing a program that can transform the input to the output. Some concepts require the use of imagination and recursion. Previously learned concepts simplify the learning of subsequent, more elaborate concepts and create a hierarchy of abstractions. We demonstrate how a robot can use these abstractions to interpret novel concepts presented to it as schematic images and then apply those concepts in very different situations. By bringing cognitive science ideas on mental imagery, perceptual symbols, embodied cognition, and deictic mechanisms into the realm of machine learning, our work brings us closer to the goal of building robots that have interpretable representations and common sense.

INTRODUCTION

Humans are good at inferring the concepts conveyed in a pair of images and then applying them in a completely different setting—for example, the concept of stacking red and green objects in Fig. 1A applied to the different settings in Fig. 1 (B to D). The human-inferred concepts are at a sufficiently high level to be effortlessly applied in situations that look very different, a capacity so natural that it is used by IKEA and LEGO to make language-independent assembly instructions. In contrast, robots are currently programmed by tediously specifying the desired object locations and poses or by imitation learning where the robot mimics the actions from a demonstration (*1*–*4*). By relying on brittle stimulus-response mapping from image frames to actions, the imitation-learning policies often do not generalize to variations in the environment, which might include changes in size, shape, and/or appearance of objects; their relative positions; background clutter; and lighting conditions (*5*).

If, like people, a robot could extract the conceptual representation from pairs of images given as training examples (Fig. 1A) and then apply the concept in markedly different situations and embodiments, then it would greatly increase their adaptability to new situations and to unstructured environments. A shared conceptual structure with humans would also simplify communicating tasks to a robot at a high level and help to improve their conceptual repertoire with further interactions.

A concept is a redescription of everyday experience into a higher level of abstraction (*6*, *7*). One way to characterize the pictures in Fig. 1 is a pixel-by-pixel description of the changes from the input image to the output, a description that will not generalize to new situations. Concepts enable a higher level of description that generalizes to new situations and ground (*8*, *9*) verbal expressions like "stack green objects on the right" with real-world referents. In contrast to the visuospatial concepts like the one in Fig. 1A that are easy and immediate even for children (*10*), numerical concepts like the one shown in Fig. 1E are neither easy nor immediate for people. The concepts that are easy and immediate, and form the basis of common sense in humans, are a very small subset of all potential concepts.

Here, we hypothesize that human concepts are programs, termed cognitive programs (*11*), implemented on a biased Turing machine (*12*) [e.g., a "Human Turing Machine" (*13*)] whose architectural constraints and biases are fundamentally different from the prevalent von Neumann style (*14*) architectures. Under this hypothesis, the inductive biases encoded in the architecture and instruction set of this computer explain why visuospatial concepts like those in Fig. 1A are easy and intuitive for humans, whereas the numeric concept shown in Fig. 1E is more difficult and unintuitive. In this view, concepts arise from the sequencing of elemental operations (*15*) on a cognitive computer according to a probabilistic language of thought (*16*, *17*), and their generalization to new situations arises out of skillful deployment of visual attention, imagination, and actions.

Our current work builds on several key ideas from both cognitive and systems neuroscience—such as visual routines (*18*), perceptual symbol systems (*6*), image schemas (*7*, *19*, *20*), deictic mechanisms (*21*), and mental imagery (*22*)—and brings them into the foray of machine learning. Following the ideas of perceptual symbol systems (*6*) and image schemas (*23*), we treat concepts as simulations in a sensorimotor system with imageable spatial information forming its fundamental building block (*20*). To this end, we developed a computer architecture called visual cognitive computer (VCC) and represented concepts as programs on this computer. The main components of VCC include a vision hierarchy (VH) (*24*), a dynamics model for interactions between objects (*25*), an attention controller, an imagination blackboard, a limb controller, a set of primitives, and program induction. We evaluated our architecture on its ability to represent and infer visuospatial concepts that cognitive scientists consider to be the fundamental building blocks (*20*). By building a working computational model and by evaluating it on real-world robotics applications, we brought several of these ideas, which exist purely as descriptive theories, into a concrete framework useful for hypothesis testing and applications.

Given input-output examples indicating concepts (Fig. 1A), we induced programs in this architecture and executed those programs on

Vicarious AI, CA, USA.
*Corresponding author. Email: miguel@vicarious.com (M.L.-G.); dileep@vicarious.com (D.G.)

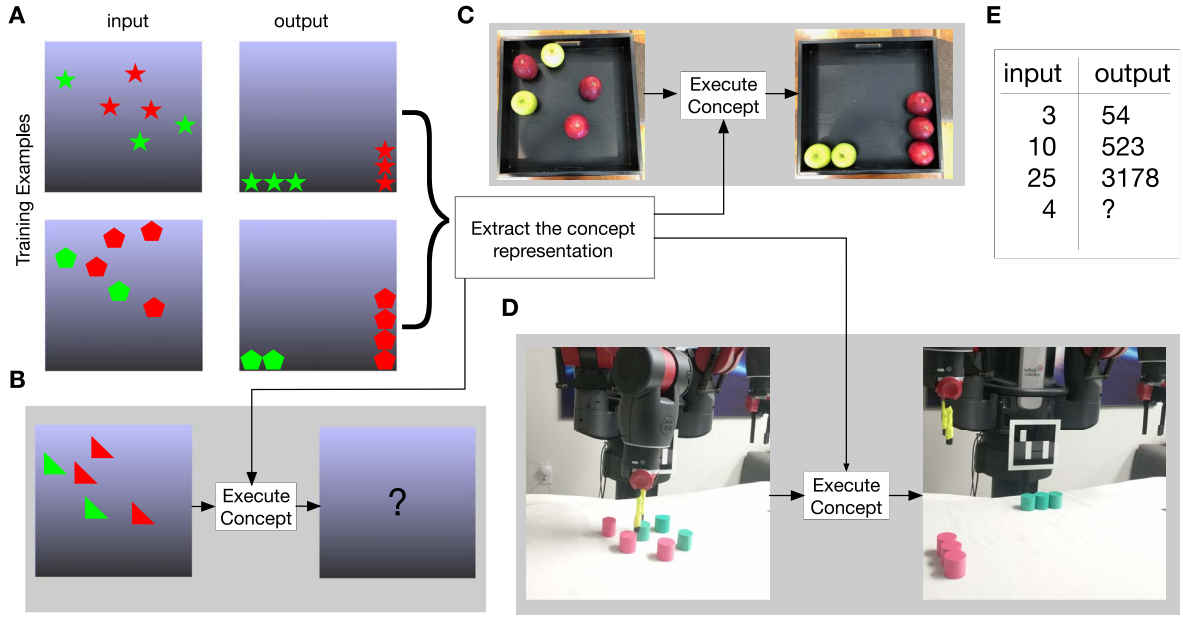


Fig. 1. People can easily understand the concept conveyed in pairs of images, a capability that is exploited by LEGO and IKEA assembly diagrams. (**A**) People interpret the concept conveyed by these images as stacking red objects vertically on the right and green objects horizontally at the bottom. (**B**) Given a novel image, people can predict what the result of executing the concept would be. (**C**) Concepts inferred from schematic images as in (A) can be applied in real-world settings. (**D**) Enabling robots to understand concepts conveyed in image pairs will significantly simplify communicating tasks to robots. (**E**) Not all concepts conveyed as input-output pairs are as readily apparent to humans as the visual and spatial reasoning tasks.

different robots to perform the desired tasks (Fig. 1, B to D). In contrast to imitation learning where a robot mimics a demonstration in the same setting, we show that the cognitive programs induced on our proposed architecture learned the underlying concepts and generalized well to markedly new settings without explicit demonstrations. Cognitive programs exhibited schematic-to-real transfer similar to the capability of humans to understand concepts from schematic drawings and then apply them in situations that look very different (*26*).

Cognitive programs on a VCC

In this section, we describe the architecture of VCC, introduce the tabletop world where it is evaluated, and provide an overview of program induction on VCC. VCC is a mechanistic consolidation of visual perception, a dynamics model, actions, attention control, working memories, and deictic mechanisms into a computer architecture. The design process for VCC started with architectural sketches provided in previous works (*11*, *13*) based on cognitive (*6*, *18*) and neuroscience (*15*, *22*) considerations. These architectural sketches provided functional requirements, rough block diagrams, and descriptive theories as a starting point but no computational models. This initial architecture was then refined and extended by codesigning it with an instruction set from the viewpoint of succinctness and completeness in program induction (*17*, *27*).

Figure 2A shows the architecture of VCC that includes the embodiment of the agent. The agent has a hand that can move objects either in the real world or in imagination and an eye whose center can be positioned within an input scene using fixation movements. The VH can parse input scenes containing multiple objects and can imagine objects, similar to the generative model we developed in (*24*) where top-down attention is used to segment out objects from background clutter. Parsed objects of a scene are stored in the object-indexing memory and are ordered according to their distance from the center of fixation. The dynamics model combined with the VH lets VCC predict the effect of imagined movements and write those results into the imagination blackboard. The attention controller is used selectively and, in a top-down manner, attends to objects based on their category or color. Top-down attention also acts as an internal pointing mechanism (*21*) to reference objects in the imagination blackboard. An external agent—a teacher, for instance—can interact with the VCC agent by showing it images and by directing its attention with a pointer (*28*).

In addition to the imagination blackboard, VCC has other structured working memories for object indexing, color indexing, and fixation history. The working memories are structured in how they represent their content, and their locality to their controllers enforces structured access; the instructions that can read from and write to specific memories are prespecified. Figure 2B lists the instruction set of VCC and their mapping to the different controllers. The VCC instruction set was heavily influenced by the primacy of objects and spatial primitives in human cognition (*29*) and by the elemental operations that have been previously proposed (*11*, *15*). See the Supplementary Materials for implementation details of the instruction set.

One critical design consideration for VCC is the ease of program induction. As an effect of having working memories that are local and specific to instructions rather than as generic registers in a von Neumann architecture, the program induction search is vastly simplified because of the fewer unbound variables in a candidate program (see fig. S1 for input and output working memory mappings of some of the instructions). Looping constructs `loop_start` and `loop_end` are constrained to loop over the currently attended objects in a scene. The instructions `set_color_attn`, `set_shape_attn`, `fixate_location`, and `imagine_object` have arguments that determine their effect. During program induction, the arguments to be used for a particular input-output pair can be explicitly searched or predicted from the inputs by using neural networks that are trained for that purpose. In addition, the arguments to the `fixate_location` command

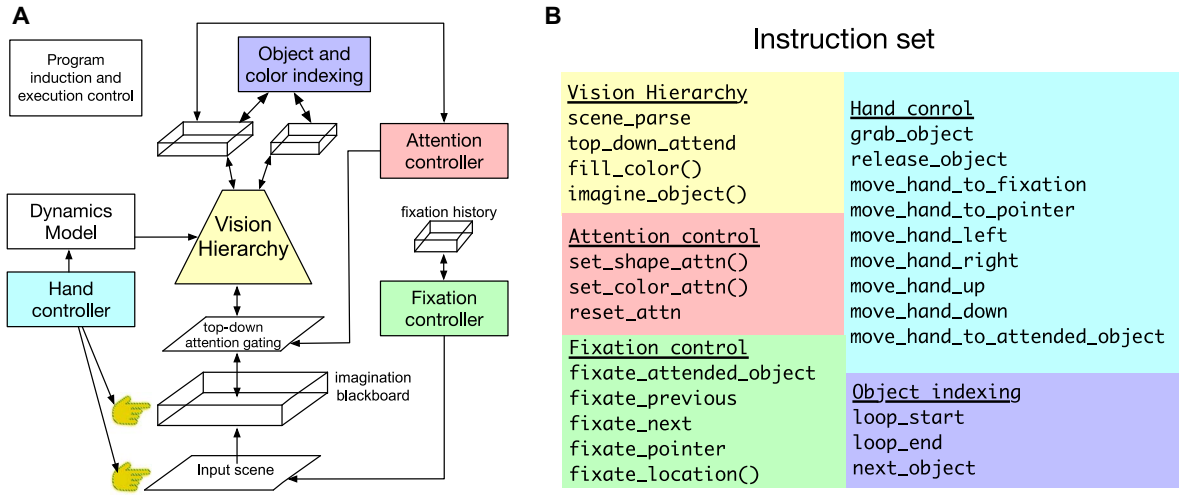


Fig. 2. Architecture and the full instruction set of the VCC. (**A**) Building blocks of VCC and their interactions. VH parses an input scene into objects and can attend to objects and imagine them. The hand controller has commands for moving the hand to different locations in the scene, and the fixation controller commands position the center of the eye. Object indexing commands iterate through the objects currently attended to. The attention controller can set the current attention based on object shape or color. (**B**) The full instruction set of VCC. Parentheses denote instructions with arguments. All concepts are represented using learned sequences of these 24 primitive instructions.

can be set externally by a teacher by the use of a pointer. A learner that takes advantage of "fixation guidance" from a teacher or accurate predictions of instruction arguments from a neural network can speed up induction by reducing the search space.

To guide the design and evaluation of VCC, we used visual concepts in "tabletop world" (TW) corresponding to the spatial arrangement of objects on a tabletop (Fig. 3A). TW allowed us to focus on imageable objects and spatial primitives that are considered to be the first conceptual building blocks (*20*) and provided a rich set of concepts to rigorously test concept representation and generalization while having simplified dynamics and perceptual requirements compared with the full complexity of the real world. Object motions in TW are limited to sliding along the surface of the table, and they stop moving when they collide with other objects or with the boundary of the workspace. How an agent generalizes its acquired concepts depends on the regularities of the world it is exposed to and on the regularities in its interaction with the world. By being a proper subset of the real world, TW enabled representation and discovery of concepts from schematic images while still having real-world counterparts. The infinite number of physical realizations of each TW concept enabled testing for strong generalization on a physical robot.

Figure 3B shows a simple, manually written cognitive program for a concept, serving to illustrate the representational properties of the VCC. The concept involves making the object close to the center touch the other object in the scene. Because the fixation is centered by default, the centered object is highlighted on the first call to `top_down_attend`. After moving the hand toward that object and grabbing it, `next_object` command is used to switch the attention to the other object. Moving the grabbed object toward the other object until collision achieves the desired goal of making the objects touch. Stopping the object when it comes into contact with another object requires access to the details of the outer contours of the object, a detail that is available only at the bottom of the VH. In contrast to architectures like auto-encoders that map a visual input to a fixed encoding in a feed-forward manner (*30*), the imagination buffer of VCC allows for the details of the object to be represented and accessed from the real world as required. As anticipated in (*6*), this allows the VH to be part of an interactive querying of the world as part of a program rather than representing a scene in one feed-forward pass for downstream tasks. In our current work, we assumed that the VH and dynamics are already learned as described in our earlier works (*24*, *25*).

The problem of learning to represent concepts is this: Given a set of input-output image pairs representing a concept, induce a program that will produce the correct output image when executed on the VCC with the corresponding input image. To solve this problem, we combined insights from recent developments in program induction (*31*–*37*) with the architectural advantages of VCC. Each program can be assigned a probability based on a model for the space of programs. To efficiently find a program for a given set of input-output pairs, our induction method relies on searching for programs in decreasing order of probability, where the probabilities are determined on the basis of generative models trained on already found programs and discriminative models that are conditioned on current inputs (Fig. 4A). We used Markov chains of instruction-to-instruction transitions (blue arrows in Fig. 4A) as generative models. This was augmented with subroutine discovery, which replaces a sequence of atomic instructions with a new instruction that can then be used in the Markov chain. Updating the generative model can be understood in the explore-compress (E-C) framework (*34*) where induction alternates between an exploration phase and a compression phase. During exploration, an existing generative model was used to guide the search of new programs, and during compression, all the programs found so far were used to update the generative model. The generative model works as an input-agnostic prior for the space of future programs given the concepts learned earlier. During search, this prior was combined with "argument predictions": predictions from discriminative models (neural networks) (*32*) about the value that the argument of each instruction will take, given that the instruction is part of the program. This prediction is specific to instructions (green arrows in Fig. 4A) and conditional on the specific input-output pairs that the program is being induced for. If a teacher is available, then fixation guidance can be used as an additional signal to predict the arguments of certain instructions. In addition, run-time exceptions generated from the VCC, represented as solid red circles in Fig. 4A, were used to prune the search space. See Materials and Methods for more details.

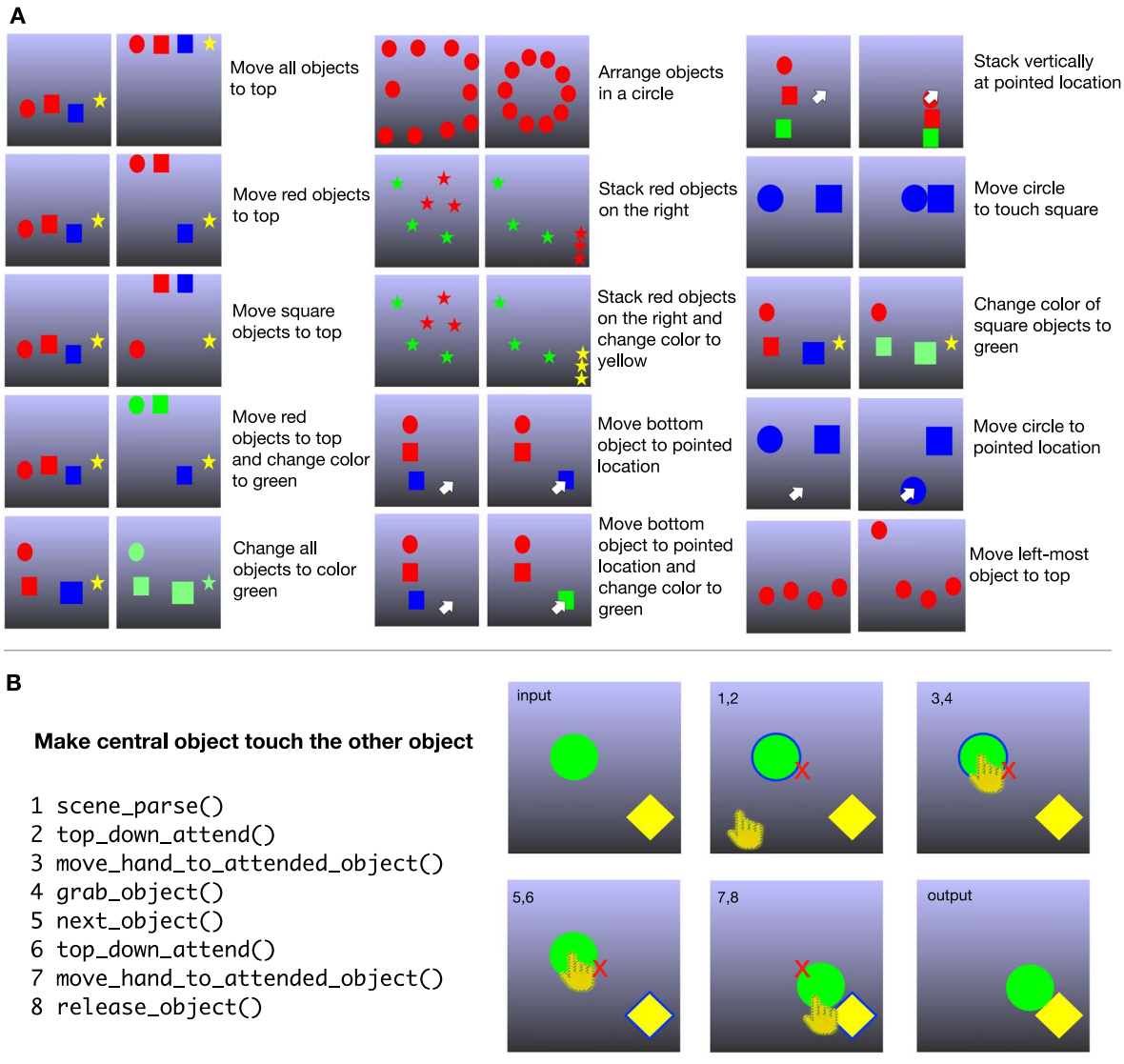


Fig. 3. Concepts and their representation as cognitive programs. (**A**) Input-output examples for 15 different tabletop concepts. In our work, we tested on 546 different concepts (see the Supplementary Materials for the full list). (**B**) A manually written program for a concept that requires moving the central object to touch the other object. The images on the right show different stages during the execution of the program, with the corresponding line numbers indicated. The attended object is indicated by a blue outline.

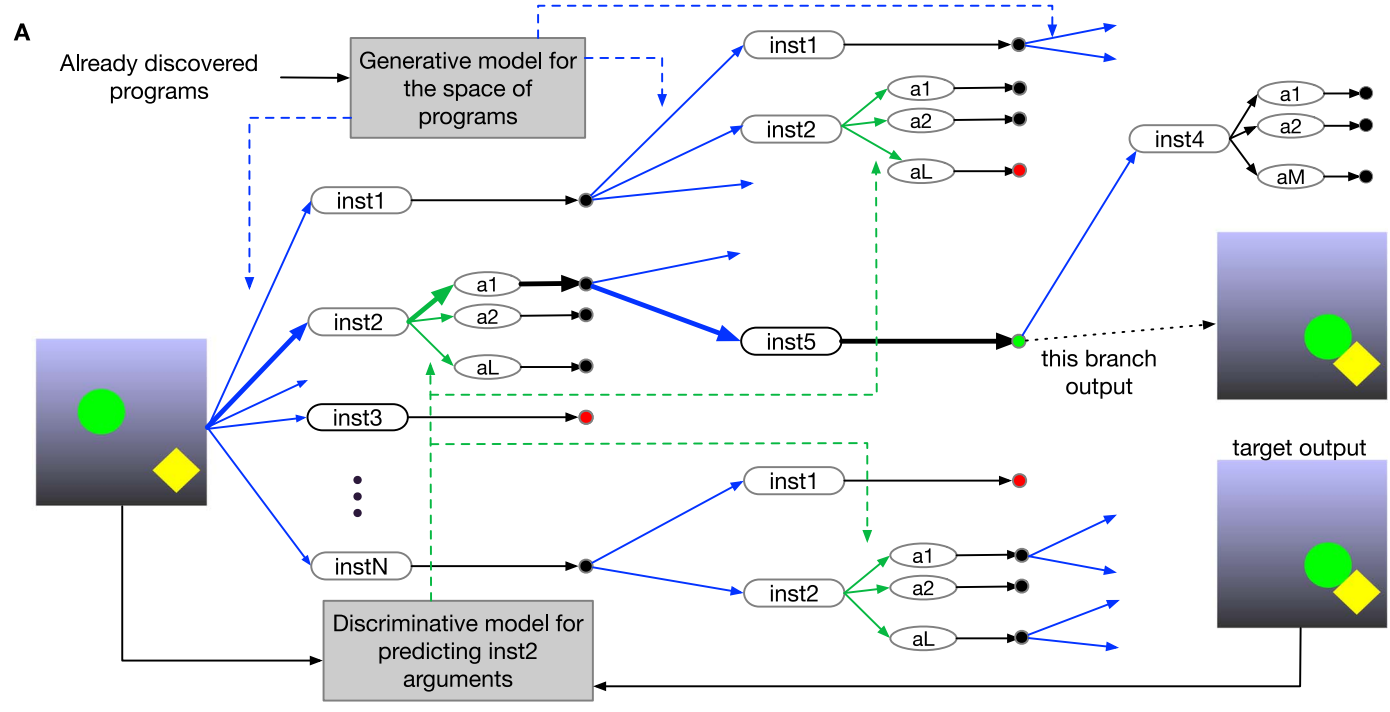
Although the VCC instructions are named to be meaningful to humans to help with interpretation, the agent is unaware of the meanings of these actions or how the different working memories and indexing mechanisms can be leveraged to represent concepts. By learning programs, the agent has to discover how to exploit the actions and properties of the VCC to represent various concepts.

RESULTS

Our experiments show that concepts could be induced as cognitive programs and that this enabled transfer from diagrammatic representations to execution on a robot without any explicit demonstration. We evaluated the performance of VCC on 546 different concepts, for which manually written programs varied in number of instructions from 4 to 23 (Fig. 4). We investigated how model-based search, input-conditional argument prediction, and fixation guidance affect the speed or learning. Using a combination of the best models, argument prediction, and fixation guidance, 535 of the 546 concepts could be learned with a search budget of 1 million programs, and 526 of them could be learned with a search budget of 0.5 million programs. The induced concepts readily generalized to settings where we varied the number and size of objects, their appearance, and the color and texture of the background. They could also be readily transferred to novel situations on multiple robots to execute tasks in the real world.

Induced programs

Figure 4 (B to D) shows three examples from the 535 induced programs that demonstrate how program induction used VCC properties to represent concepts. The first example (Fig. 4B) requires to move the left-most object to the top. The learned program used the property that VCC's object-indexing memory orders objects by their distance from the center of fixation. Although "the left-most object" is not an attribute



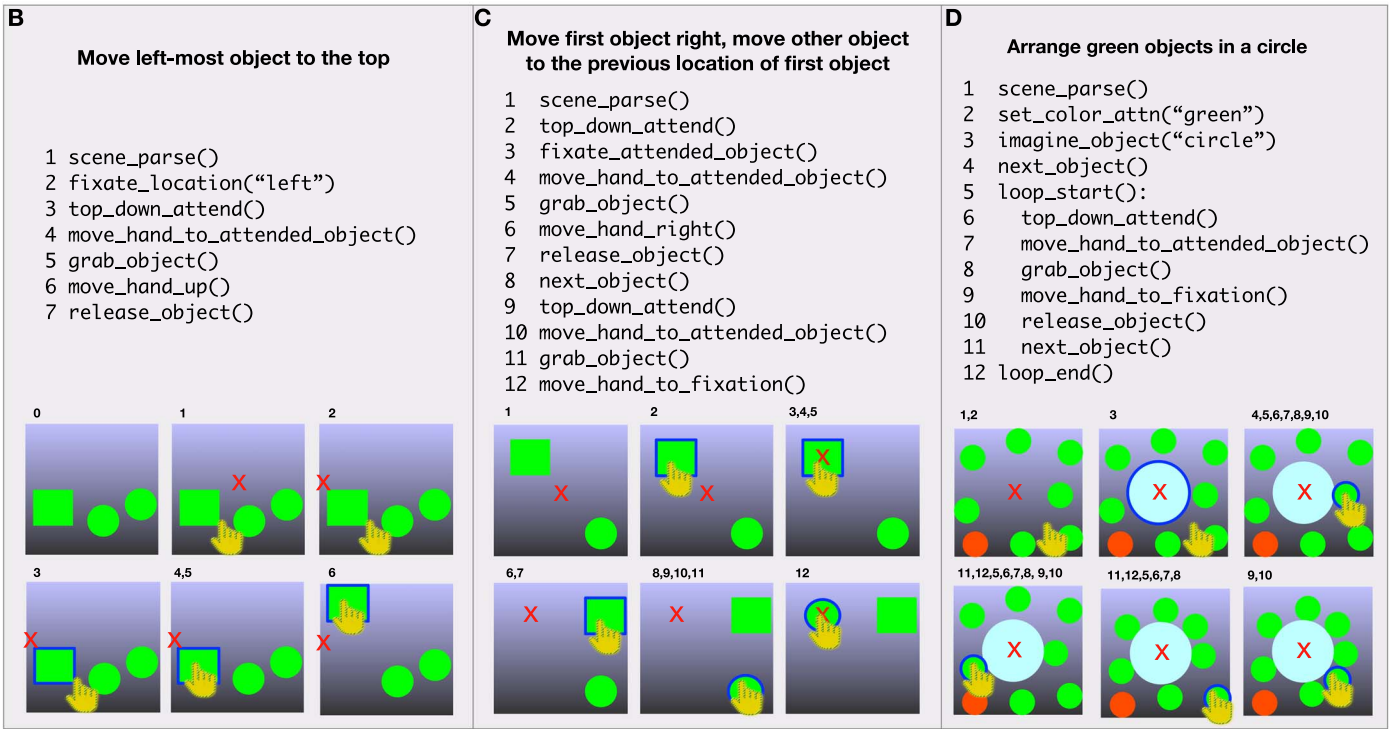


Fig. 4. Program search and discovered programs. (**A**) Program induction searches in an exponential space represented as a tree, where each node (solid circle) is a program. Blue branches are instruction-to-instruction transition probabilities modeled using a generative model, and green branches are instruction-to-argument probabilities predicted using discriminative neural nets trained on input-output images. The probability of a program depends on the weights of the branches leading to its node. Solid red circles are programs that generated an exception in the VCC, and the green node is a correct program. (**B** to **D**) Three examples of discovered programs and visualizations of their execution steps. Digits next to the visualizations correspond to program line numbers.

directly encoded in VCC, that concept was represented as a combination of other primitive actions.

Figure 4C shows an example where induction discovered the use of a deictic mechanism. The concept required moving the central object to the right edge and moving the other object to the position previously occupied by the first object. Achieving this requires holding the position of the previous object in working memory. Induction discovered that the fixation center can be used as a pointer to index and hold object locations in memory. The discovered program moves the center of the eye to the first object after attending to it and grabbing it. Because the eye fixation remains at the initial position of the first object when the object is moved away, the second object can be moved to the correct position by simply moving it to the fixated location. The concept of arranging small circles into a big circle (Fig. 4D) requires the agent to imagine an object that does not exist (the big circle) and then push other objects toward this imagined object while it is maintained in the working memory. This is an example where teaching by demonstration is likely to face challenges because the intermediate stages of that demonstration would provide only scant cues about the final goal of arranging objects in a circle.

Induction dynamics

We tested the efficacy of program induction under a wide variety of settings. For model-based search, we compared an order-0 model that only considers the relative frequencies of instructions to an order-1 model that learns the transition probabilities of instructions in programs. In each case, we tested the effect of learning subroutines. We tested the effect of combining the order-1 model with neural network-based predictions for the arguments of each instruction. We also tested the effect of fixation guidance where pointing action by a teacher was used to set the fixation location in the learning agent.

Iteratively discovering the programs by modifying the model that is driving the search process and augmenting that model with input-conditioned argument predictions are both important for discovering the complex concepts with longer programs. Figure 5 (A to D) shows the length distributions of the programs induced after each E-C iteration for different settings, for a search budget of 1 million programs. The histogram of the number of programs of different lengths is plotted along with a ground-truth distribution. (The ground truth is derived on the basis of manually writing programs for each of the concepts. The discovered program for any particular concept need not match the ground-truth program in implementation or length). When argument prediction was not available, program induction relied on searching exhaustively in the argument space. Without argument prediction, no programs with a length of more than 12 instructions were induced, whereas with input-conditioned prediction of arguments programs with a length of up to 18 instructions were induced. Overall, without argument prediction, only 150 of the 546 concepts were induced, whereas argument prediction and subroutines enabled the discovery of 526 of 546 concepts. Iterative refinement of the model using the discovered programs at each stage resulted in more programs being discovered in the next iteration of search. Figure 5 (A to C) shows the length distributions of programs discovered after each E-C iteration. The distributions of the discovered programs shifted right with each iteration, successively discovering longer programs that were missed in the previous iteration. The progression of concepts learned during multiple E-C iterations demonstrates the advantage of learning to learn (*38*) in program induction.

A teacher can help the agent learn faster by guiding the agent's attention by pointing. We tested the effect of having a teacher guide the fixation of the agent for concepts that require using the `fixate_location()` function similar to the joint attention mechanism hypothesized in cognitive science (*28*). This was achieved by setting the argument of the `fixate_location()` function to the location pointed by the teacher. In contrast to other instructions, the arguments for the `fixate_location()` function were not predictable from the input-output image pairs, and program induction relied on searching the argument space in the absence of any other guidance. The effect of offering fixation guidance was most significant at search budget of 0.4 million, as shown in Fig. 5E. In this setting, using fixation guidance increased the number of discovered programs from 148 to 524. Although fixation guidance from a teacher was applied here in a very limited setting, this shows that prespecified joint attention mechanisms with a teacher hold promise for significantly speeding up induction.

Overall, using model-based search using an order-1 model and argument prediction were the most important factors that determined how quickly the programs could be learned. Figure 5 (E to H) shows the effect of various factors as the search budget was varied from 0.25 to 3 million programs. When an order-1 model was used, induction of subroutines played a smaller role in the induction of new programs: Although subroutines that reduced the description length were obtained after each compression iteration, they provided only a modest help in the future discovery of programs. To give a sense of the required computational effort, we note that searching 1 million programs took around 10 min when using a single core.

Modeling the sequential relationship between the instructions of learned concepts significantly helped with program induction, compared with modeling just the instruction frequencies. We tested an order-0 model that ignored the sequential dependencies and an order-1 Markov model between the instructions as the models used in the E-C iterations. Of 546 concepts and with a search budget of 1 million programs, the order-0 model was not able to induce any new concept when subroutines were disabled and only 7 new concepts when they were enabled (*39*). In contrast, an order-1 model was able to learn 525 of the 546 concepts for the same search budget (Fig. 6B). Figure 6A shows the transition matrix of the order-1 Markov model after iterations 0, 2, and 4. The transition matrix learned additional relevant modifications, with each iteration that enabled more concepts to be learned in future iterations.

Most concepts were learned correctly with just a few input-output examples and generalized well to markedly different images depicting the same concepts. Figure 6C shows the learning curve for concepts discovered in the first E-C iteration (107 concepts, including 16 bootstrapping ones). The learning curve flattens out after five examples. Figure 6D shows an example of the confusion caused when the number of examples is fewer. Although the concept required moving the yellow triangle to the middle, in the first two examples, this could be achieved just by moving to the middle whichever triangle was closest to the center. The generalization of the induced concepts to new situations was tested using concept images generated with different numbers of objects and visual transformations, examples of which are shown in Fig. 7A. The test images were generated by varying the backgrounds, object foregrounds, number and kinds of objects, and their placements within the image. For each concept, we tested on 50 different test images generated by sampling from these parameters. More examples of test images are shown in fig. S4. All the induced concepts generalized to a 100% of these new settings.

Transfer to robots

Induced concepts transferred readily from the schematic inputs to the real-world execution on two different robots: a Baxter from Rethink

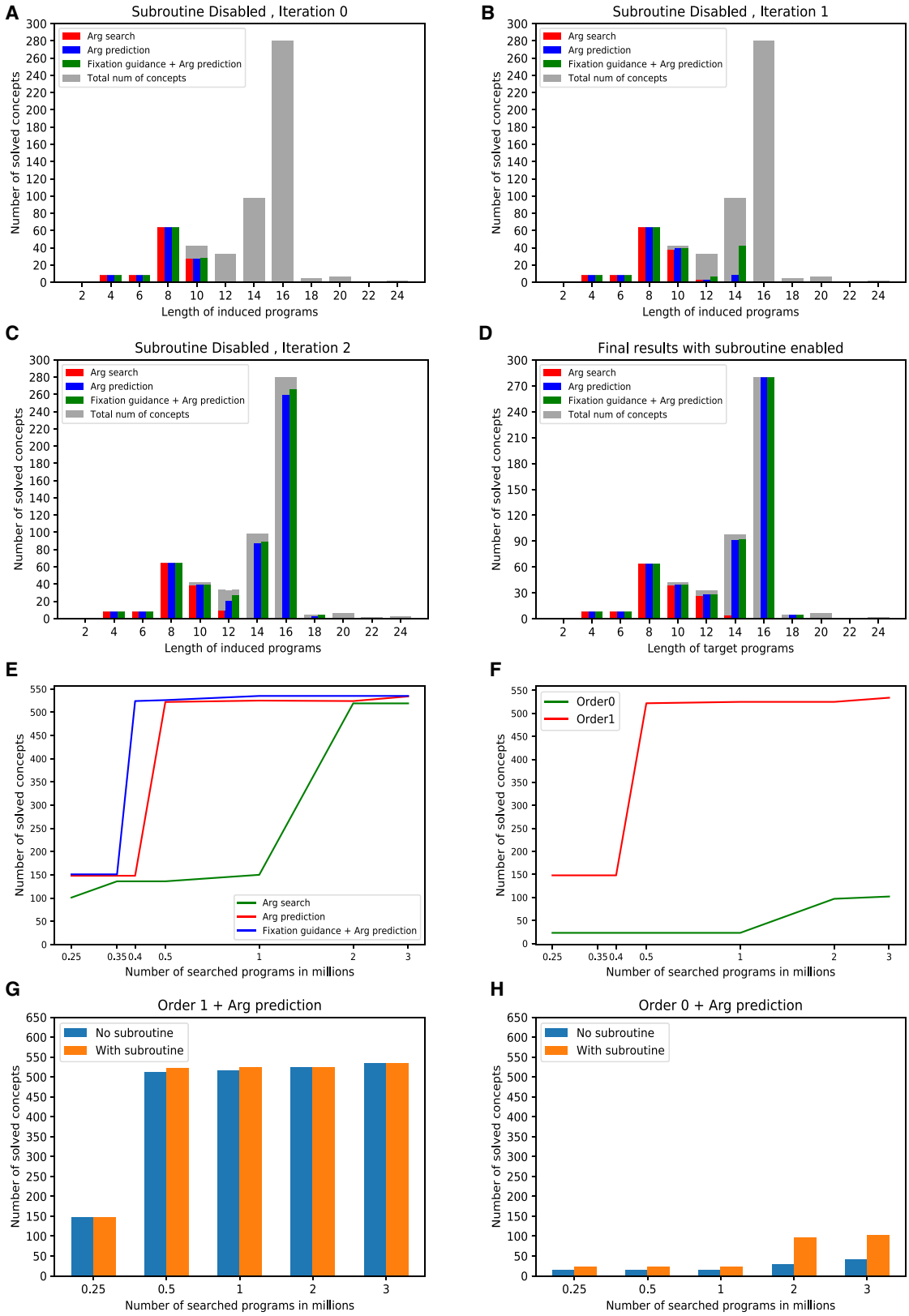


Fig. 5. Program induction details. (**A** to **C**) Length distribution of induced programs for the first three E-C iterations. *X* axis bins correspond to program lengths (number of atomic instructions). The gray bars represent the total number of programs of that length according to a set of manually written programs comprising all concepts. (**D**) Distribution at the end of all iterations. (**E** and **F**) Number of induced programs for different search budgets and for different model options. (**G** and **H**) Effect of subroutines.

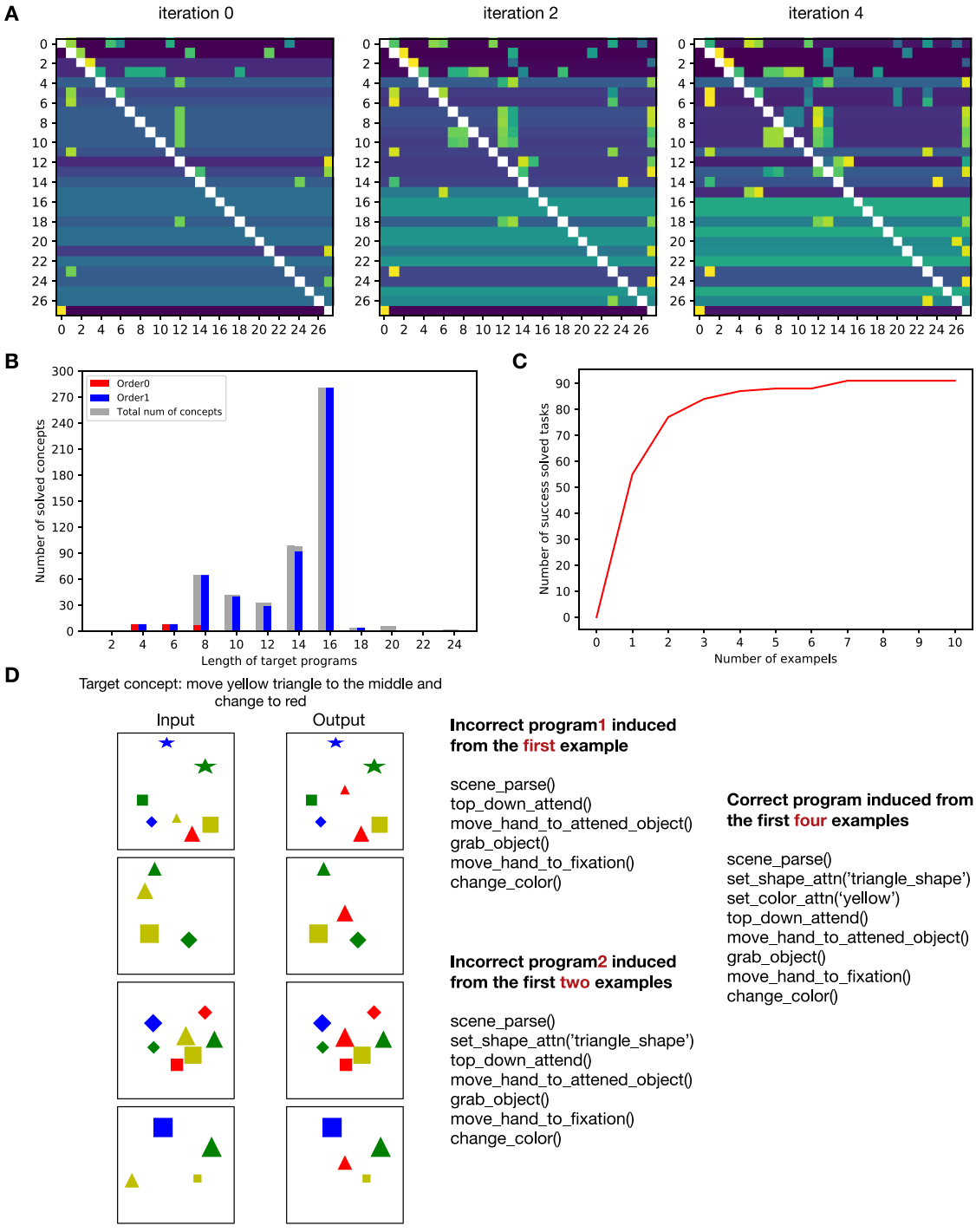


Fig. 6. Program induction and generalization. (**A**) The instruction-to-instruction transition matrix after different E-C iterations. (**B**) Length distribution of programs induced using order-0 versus order-1 model. (**C**) Training curve. Most concepts are solved with just a few examples. (**D**) An example showing wrongly induced programs when only three training examples from a concept are presented, where accidental patterns in the data can explain the examples. In this case, the correct concept was induced with four examples.

Robotics and a UR5 from Universal Robots. The primitive instructions for perception and action, such as capturing the input scene and controlling the hand, were implemented using the respective robots' interface, while the rest of the VCC instructions remained the same. We tested the transfer of five different induced concepts on the two robots, and each concept was tested in several different scenarios by changing the number and types of objects, their positions, the number of distractor objects, and the appearance of the backgrounds. In each case, the initial state of the test scenarios was constructed such that running the induced program on the input image would produce the right output in VCC's imagination.

Figures 7 and 8 show the robots at different stages of the execution of the concepts for a sampling of the settings we tested on. The concepts

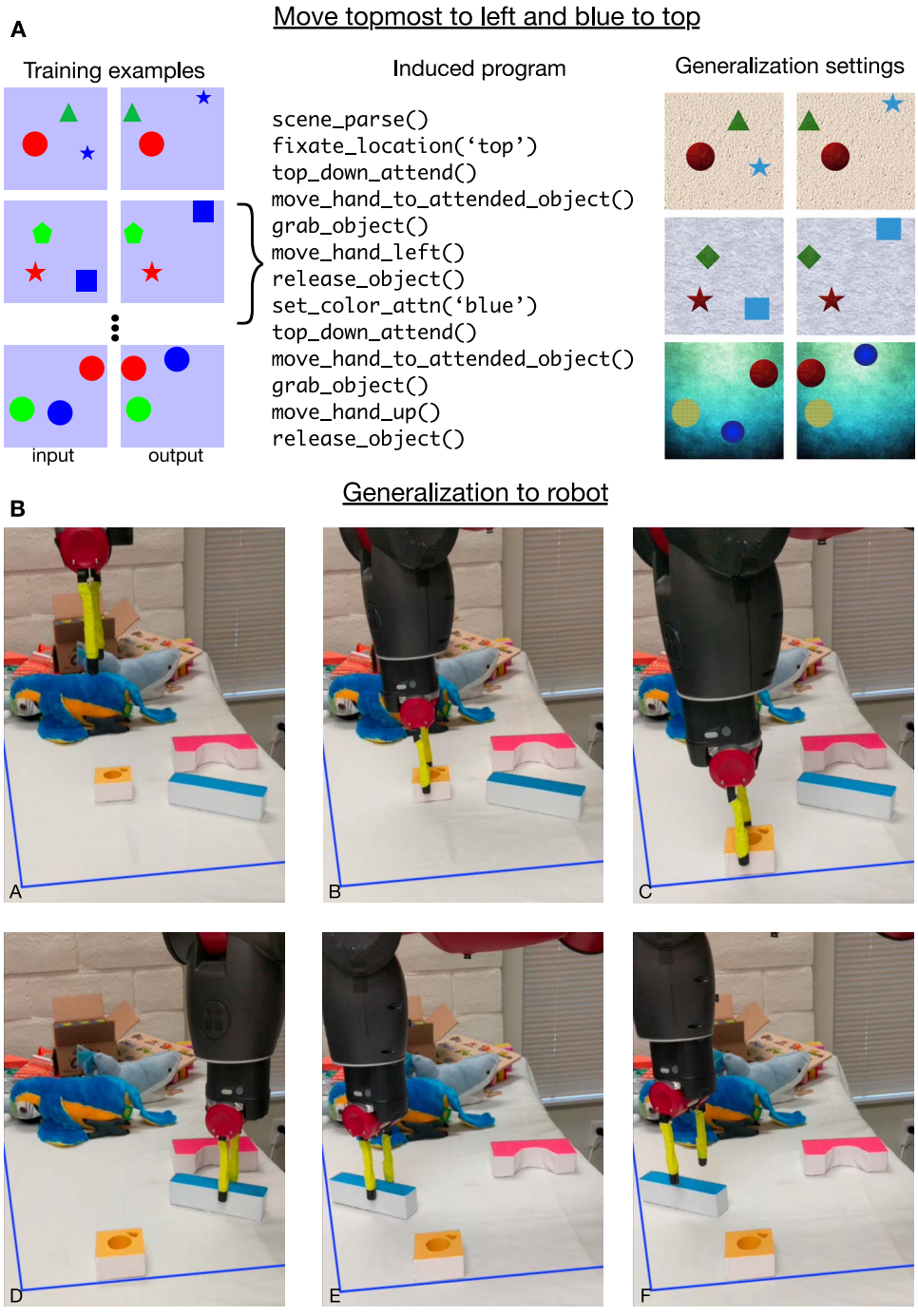


Fig. 7. Generalizing to new settings and to the real world. (**A**) Training examples and induced program corresponding to the concept "move topmost to left and blue to top." The right columns show different test settings to which the program generalizes. The test settings are all very different from the training setting except for their conceptual content. (**B**) The concept in (A) executed on Baxter robot with very different objects compared with the training setting. Different stages in the execution are shown.

are executed correctly despite the variations to the number of objects, their sizes, positions, and variations in the background, as long as the information relevant for the concept is present in the input. In Fig. 8A, we show the stacking concept executed in three different settings. The different stacking concepts were executed correctly with the objects properly in contact in their final states, even when the sizes and types of objects were significantly different. Out of the total test scenarios that were tried, the robot UR5 succeeded in executing the concepts in more than 90% of the trials. The failures were caused by the objects slipping out of the gripper during a grasp or during placement. On Baxter, the success rate was lower (at about 70%) due to a blurry camera, a less effective gripper, and imprecise motion in our aging robot.

We also show how a complex concept that could not be induced directly could be broken into two different concepts and executed consecutively on the robot to achieve the more complex task. The final task requires moving the yellow objects on the table to the left and the green object on the table to the right, which can be achieved as a combination of the first set of movements with the second. We show an application where the concept can be used to separate lemons from limes (Fig. 8B). The reader is encouraged to view the robot demonstration videos available as the Supplementary Materials.

DISCUSSION

Getting robots to perform tasks without explicit programming is one of the goals in artificial intelligence and robotics. Imitation learning, a predominant approach to this end, seeks to teach the robots via demonstration. To demonstrate the actions required for a task, a robot could be physically guided or operated remotely (*1*, *40*). Robot operation is very time consuming, so one-shot approaches that try to learn executable policies from a single demonstration have been developed (*2*, *4*, *41*). The main drawbacks of imitation learning approaches are their focus on rote mimicry without understanding the intent of a demonstration and their tendency to latch onto surface statistics and idiosyncrasies of the demonstration (*19*) rather than its conceptual content, resulting in limited generalization to new settings. Recent work (*3*) has sought to build an intermediate representation from a demonstration that would allow generalization to more settings but still relied on having a demonstration available for each variation of a task, in addition to having a large annotated background set for the family of variations.

By focusing on the discovery of conceptual understanding from pairs of images, our work is very different from the traditional setting of imitation learning. No demonstrations are available, and the agent has to discover the conceptual content and transformation that are

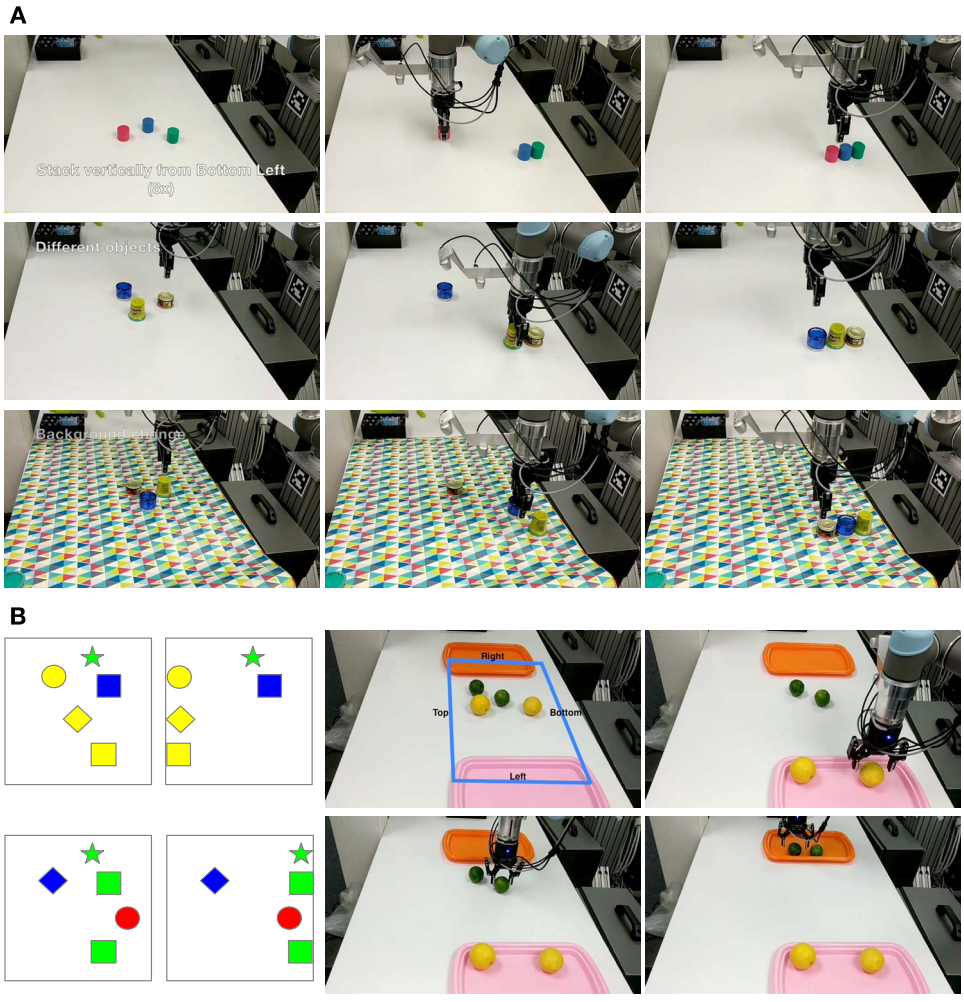


Fig. 8. Learned concepts transferred to different real-world settings. (**A**) Each row shows the starting state, an intermediate state, and the ending state for three different execution scenarios for a concept that requires stacking objects on the bottom left. The middle row shows execution on different objects, and the bottom row shows execution on a different background. (**B**) Execution frames from an application that separates limes from lemons. This task is achieved by the sequential composition of two concepts. (Left) The two concepts used (top and bottom) and (right) execution of these concepts in sequence to achieve the task.

represented by the image pairs. Moreover, the discovered representations need to transfer to very different settings, including markedly different visual appearances and different robot embodiments. Such transfer requires visual analogy making (*42*, *43*) as opposed to rote imitation. Although we took inspiration from earlier works on learning programs to describe the contents of images and videos (*38*, *44*, *45*), those works focused on settings where the content to be described was fully observable in the images or video demonstration. Our setting is very different and significantly more challenging because of the need to discover hidden variables underlying the concepts, the number of steps involved in going from an input image to the output, and the need for strong schematic-to-real generalization in settings significantly different from the training distribution.

The sequential and programmatic nature of conceptual representations has been well recognized in cognitive science and neuroscience, with Ullman's seminal paper on visual routines (*18*) and its implementations (*46*–*52*), and in prevalent cognitive architectures (*53*, *54*) that use sequential representations. Although sharing the motivations of sequential representations and chunking, our work is a significant departure from the traditional cognitive architectures of ACT-R and SOAR in emphasizing the perceptual, motor, and interactive nature of concept representations. In having a VH that is not a passive feed-forward transducer, but an active system that can be manipulated and reconfigured (*11*), our architecture follows the guidelines of perceptual symbol systems (*6*) rather than pure symbol manipulation. VCC and cognitive programs can also be thought of as a concrete computational instantiation of image schemas (*55*), an influential idea in cognitive science that emphasizes the grounded and embodied nature of concepts. The programs that we learn on VCC can be considered "patterns of sensorimotor experience" that constitute a concept, forming the basis for communication and grounded language (*56*). Our work is also consistent with the idea of a probabilistic language of thought (*57*) where complex representations are formed from small repertoire of primitives (*16*, *17*). Representing concepts as programs makes them naturally compositional as required for a language of thought (*57*), and programs provide explainability that has been lacking in current black-box policy learning methods.

The tremendous success of deep learning (*58*, *59*) along with discoveries of its limitations has rekindled the age-old debate about innate biases versus tabula rasa learning (*60*). Bucking the recent trend of tabula rasa learning with large training sets, our work here focused on learning programs on a biased computer architecture whose biases were inspired by cognitive science and neuroscience and by computational considerations. We carefully considered the question of what is a good architectural starting point (*26*) for the learning of concepts: What should be the design of the processor (*13*)? What should be the elemental operations (*22*)? We believe that the seeming arbitrariness and ambiguity of the starting point is part of the challenge in bringing these ideas into a computational model and need to be confronted directly (*61*). We tackled this challenge by treating this as analogous to the design process of a microprocessor where choices need to be made regarding the nature of registers, memory, and instruction set. As described earlier, cognitive and neurosciences provided significant guidance, which were then refined from the view point of program induction. Although it is unlikely that all the detailed choices we made in this first iteration are the ones that will give rise to human-like concepts, many aspects of VCC regarding the interaction of VH, imagery, working memory, attention, action, and program induction are likely to remain in future iterations. Through further theorizing and experimenting the design of the VCC will be expanded and refined. We believe that this iterative

refinement of architectural priors, inductive biases, and learning algorithms is an essential part of building systems that work like the human mind (*26*).

A particularly important design choice is that of VH and top-down attention. Rather than follow the prevailing machine-learning view of treating vision as re-encoding the input for "downstream tasks" (*62*), we treat the VH as a mechanism for structured interaction with the world. In this view, the encoding of the world is not just at the top of the hierarchy, and all details need not be represented at the top of the VH in a loss-less manner as in deep neural network generative models. The VH in our model is lossy, and the details can be accessed at the input on demand, consistent with neuroscience ideas of using the primary visual cortex as a high-resolution buffer for mental imagery (*22*) and with the ideas of selective tuning of attention (*63*, *64*). In this design, the pattern of accessing the detail at the bottom, or the more abstract representation at the top, becomes part of the representation of the concept. Top-down attention also serves to achieve binding (*65*)—for example, between the color and shape of an object—by using attention as an internal pointing mechanism (*21*).

Many of the existing datasets that measure conceptual and abstract reasoning have drawbacks that prevent them from being used in a study for acquiring and representing concepts as sensorimotor programs. Raven's progressive matrices (RPMs) (*66*) are often used as a test for conceptual representation. Instead of using RPM, we chose to use TW because the properties of the world that give rise to generalization are systematic and well understood. In contrast, in RPMs, the source of generalization can encompass the full experience of a young adult, including all the generalizations that arise from a fully developed VH that can reason about occlusions and transparency. Standard RPMs are also restricted to being evaluated as a multiple-choice question. TW was also inspired by the tasks introduced in (*21*) for deictic mechanisms but goes beyond those in terms of complexity, analogy making, and transfer to real-world execution. In contrast to datasets that measure pixel-accurate image reconstructions in simplistic settings (*67*), use of the TW recognizes the schematic nature of concepts (*20*) and enables the evaluation of sensorimotor representations for their generalization to settings that are different from the training distribution, including different real-world settings involving robots.

One common approach in program synthesis is to combine a domain-specific language with a context-free grammar, such that the whole space of syntactically valid programs is derivable (all programs that are generated would compile, and all programs that solve a task would be generated with nonzero probability) (*20*, *68*). To achieve this, a popular choice is to use functional programming with type checking, which guarantees successful execution of programs generated according to the grammar. In contrast, we found that an imperative programming language was more suited for our purpose and subjectively more adequate to describe the thought process associated with a concept. This does not guarantee error-free execution when sampling from a Markov model, and some programs are rejected: The machine itself becomes part of the model, conditioning on valid programs (as in rejection sampling) and effectively pruning the search space. We took inspiration from the recent work (*33*) in bringing the machine itself into program synthesis.

We are excited about the future research directions that our work opens up. A richer set of primitive instructions that supports an interplay of bottom-up and top-down attention and uses the part-based representation of the VH could enable a wider variety of concepts. Perhaps some of the primitives could be learned by having an agent interact with the environment using mechanisms we elaborated earlier (*69*). The use of joint attention by pointing can be expanded in scope to direct top-down attention, not just fixation. Attributes of an object—width, height, number of corners, etc.—could themselves be represented as sensorimotor programs that are learned with experience in the real world but evaluated purely in imagination during execution. Expanding the dynamics model to include three-dimensional (3D) objects and combining it with occlusion reasoning and surface representation abilities of the VH could result in a large number of real-world concepts being learned as cognitive programs.

MATERIALS AND METHODS

Program induction

We are provided with a set of collections of input-output pairs of images, with each collection corresponding to some unknown concept, and we wish to infer the programs that describe each of those concepts. A naïve way to perform this inference is via brute force, but this becomes unfeasible with increasing program lengths. However, it can be useful as an initial step to discover the simplest of concepts to form a bootstrapping set. To discover more sophisticated programs, we fitted a probabilistic model to the programs in the bootstrapping set and then used that model to guide the search starting from the most probable program and searching in the order of decreasing probability. Once some effort threshold was hit (for instance, number of programs considered), we collected the found programs, refitted the probabilistic model, and repeated. This approach is referred to as the E-C framework in (*34*), where compression stands for fitting a probabilistic model to data. The key for this approach to work properly is the probabilistic model.

We modeled programs (both instructions and arguments) as an observed Markov chain. The model for the instructions is learned from the already discovered concepts, whereas the emission model is conditional on the input-output pairs of examples and is learned separately. As we will see next, the whole induction process depends only on two free parameters: a modeling parameter ε (the pseudocount) and an exploration parameter M (maximum number of explored nodes).

The probabilistic model

We start by considering programs as a sequence of instructions, without arguments. A program x is a sequence of atomic instructions $x = [x_1, x_2, \ldots x_L]$, where the last instruction is always $x_L = e$, a special end-of-program marker. The probability of a single program is

$$\log p(x) = \log p(x_1) + \sum_{i=1}^{L-1} \log p(x_{i+1}|x_i)$$

where $p(x_{i+1}|x_i)$ is the transition probability from instruction x_i to instruction x_{i+1} and $p(x_1)$ are the initial probabilities. To compute the probability of multiple programs $\{x^{(i)}\}$ that we consider independent, we can just add the log probabilities of each of them. We can express this compactly by defining $X \equiv ex^{(1)}x^{(2)}\ldots$ as a sequence that simply concatenates the programs (in any order) and prepends an end-of-program marker. Then, the joint probability of multiple sequences is simply

$$\log p(X) = \sum_{i=1}^{N-1} \log p(X_{i+1}|X_i) \quad (1)$$

where N is the total length of all the programs combined, including the initial marker. From a compression perspective, $-\log p(X)$ corresponds to the description length (in nats) of the collection of programs under this model. Parameter fitting for this model amounts to determining the transition matrix T_x, where $[T_x]_{rs} = p(s|r)$. We use the maximum likelihood estimator with a small pseudocount ε to avoid overfitting.

We can further enhance the model using subroutines. Subroutines are sequences of instructions and can be incorporated into the model by adding a dictionary D with subroutine definitions and allowing instructions to index not only atomic instructions but also subroutines. A program with atomic instructions x can now be expressed in compressed form c by identifying the subroutines it contains and replacing them with a single instruction containing the appropriate subroutine call. The joint probability of all the programs X, its compressed representations C, and the subroutine dictionary D is then

$$\begin{aligned}\log p(X, C, D) &= \log p(X|C, D) + \log p(C) + \log p(D) \\ &= \log p(C) + \log p(D)\end{aligned} \tag{2}$$

where the last equality follows from X being deterministically obtained from C and D, and therefore, $\log p(X|C,D)$ equals 0 for valid expansions X of C. The two terms in the right-hand side (r.h.s.) can each be encoded as concatenated sequences (as we did for X) and computed using Eq. 1.

To fit this model for a given X, we maximize Eq. 2 with respect to the transition matrix T_c (which is shared for both programs and subroutines), the compressed representation C, and the dictionary D. This joint optimization is, in general, a difficult problem. We opt for a greedy method: We consider the increase in $\log p(X, C, D)$ that inserting each new possible subroutine in the dictionary (and updating C and T_c accordingly) would produce and insert the one that achieves the maximum gain. Then, we repeat the process until no positive increase is achievable, adding one subroutine at a time. We only consider as potential subroutine sequences of instructions that appear multiple times in the programs. In contrast to maximum likelihood estimation, the maximum a posteriori estimation of D includes the prior and provides a trade-off in which subroutines are only deemed useful if they appear often enough.

We now turn to modeling the arguments of the instructions of a program. Each program x has an accompanying set of parameters $y = [y_1, y_2, \ldots, y_L]$ of the same length as x. Thus, each instruction has exactly one parameter, which can take one value among a discrete set of choices. Those choices are different for each instruction and are given by the syntax of the language. For a given dictionary, the probability of a full program (including now arguments) is

$$\begin{aligned}\log p(y, x, c|D) &= \log p(x|c, D) + \sum_{j=1}^{L_c-1} \log p(c_{j+1}|c_j) \\ &\quad + \sum_{i=1}^{L-1} \log p(y_{i+1}|x_{i+1}) \\ &= \log p(x|c, D) + \sum_{j=1}^{L_c-1} \left[\sum_{h=1}^{n_{j+1}} \log p(y_{j+1h}|x_{j+1h}) + \log p(c_{j+1}|c_j) \right] \\ &= \sum_{j=1}^{L_c-1} \log p(y_{j+1}|c_{j+1}) + \log p(c_{j+1}|c_j)\end{aligned} \tag{3}$$

where we have collected all the arguments that are used in a subroutine c_j into a single variable $y_j = [y_{j1} \ldots y_{jn_j}]$ and removed $\log p(X|C,D) = 0$ (due to determinism) from the equation. We have already estimated all the quantities in the above expression except for the conditional probability of the arguments $\log p(y_i|x_i)$ of a given atomic instruction, which will be described in a subsequent section on argument prediction.

The exploration

To discover the concept that explains a particular input-output collection, we start by expressing the probability of a compressed program as a Markov chain

$$\log p(z) = \sum_{j=1}^{L_c-1} \log p(z_{j+1}|z_j)$$

where $z_j \equiv (c_j, y_j)$ and the transition probability T_z can be easily derived by combining T_c and $p(y_j|c_j)$. We thus have a Markov model over "program portions" z_j, which can be either individual instructions or subroutines.

The Markov model induces an exploration tree: Each node has as children all the possible program portions and is connected to them through arcs with cost $-\log p(z_{\text{child}}|z_{\text{parent}})$. Each node corresponds to a program, and its description length can be obtained by adding the weights of the arcs on the path from the root. A best-first traversal of the tree (always expanding the nodes that have less accumulated cost from the root) visits the programs in order of decreasing probability. Each visited compressed program z can be expanded into its atomic version x and run on the VCC, checking whether the produced outputs match the provided ones. We stop the process after we find a valid program (the concept is discovered) or the number of visited nodes exceeds some effort parameter M.

To alleviate the memory demands of best-first search, we use iterative deepening where depth-first search (DFS) is run with a limit on the description length, the limit is gradually increased, and the process is repeated. The nodes (programs) visited on each iteration that were not visited in the previous iteration are then run on the VCC. This visits the nodes in approximately a best-first order: Within each search bracket, the node ordering is arbitrary, but the brackets are ordered. The smaller the brackets, the tighter the approximation to best-first search. VCC states are cached during the DFS traversal to prevent redundant instruction executions, and the successors of any node that produced an "invalid" execution on VCC are pruned away from further search.

Because the number of explored programs is relatively low, we first run a best-first search that identifies the maximum description length of the shortest M programs (without running them), and then, we run DFS using the identified description length as the cutoff. During the DFS, we run the programs, taking advantage of the optimizations described in the previous paragraph. The DFS is run until completion even if the sought-for concept is found earlier, because it does not guarantee that the shortest description of the concept will be found first.

Argument prediction

The differences and similarities between the input and output images that are used as examples for each concept provide information about what to pay attention to. We use this information to predict the argument for the following three functions: `set_shape_attn`, `set_color_attn`, and `fill_color`. Given that the argument of each function can take several potential values and each function

can appear multiple times in a program with its argument taking a different value, this becomes a multilabel problem. We build a logistic regression model for each function and argument pair.

To get training data for argument prediction, we enumerate all valid programs up to length 6 (including `scene_parse`), where valid programs are those that can be executed on input images without any failure. From the set of valid programs, we filter out uninformative programs whose output images are the same as the corresponding input. We also remove programs where the presence of an instruction with its assigned argument does not lead to different output compared with the program without that instruction.

For each program, we generate 10 examples. Each example is converted to a 3D binary array A with shape (21, 15, 15), where 21 is the number of input channels and 15 corresponds to the height and width of the discretized images. The first 10 channels are based on the input images, as shown in the first column of fig. S2. Each element is set to 1 or 0, depending on whether the feature associated with that channel is present at that location or not, respectively. The next 10 channels are based on the difference between input and output images, both using the same binary encoding, which results in the elements of these channels having −1, 0, and 1 as possible values. This is shown in the third column of fig. S2. The difference between output and input highlights what has changed between both images. For example, when the green color disappears, a −1 value is registered in the third column at the corresponding location and for the green color channel. In contrast, the blue square remains in the same position; thus, it becomes zero when we subtract the input from the output. The last channel summarizes the differences across the 10 previous features (channels). Thus, an element corresponding to row r and column c of channel 21 is computed as $A_{21rc} = \sum_{f=11}^{20} |A_{frc}| \geq 1$. That is, a value of 1 at a given position means that the object at that position has changed regardless of whether it was added or removed. This feature combines color change and movement change into one single indicator.

We use a convolutional neural network (CNN) to capture spatial invariance, because the object experiencing a change can be anywhere in an image. Figure S3 shows the architecture of the model. To capture the similarities among examples, we sum the max pooling results from all examples and feed it into a sigmoid function. Given that the last layer is simply performing a summation, we only need to train the convolutional weights of the CNN. The model is trained using the ADAM optimizer and L1 regularization with a weight of 0.01. There are other two functions with arguments that are not supported by this model and for which we do not perform predictions: One of them is `fixate_location` and the other one is `imagine_shape`.

We also use the result of argument prediction to predict the existence of an instruction in a concept. Specifically, if, for a given instruction, the sum of probabilities of all the values that its argument can take is below 0.55, then it is assumed that the instruction is not present in the concept and excluded from the search process.

Execution on robots

We tested the transfer of induced concepts to real world by executing programs on two different robots in different settings: a Baxter robot from Rethink Robotics and a UR5 from Universal Robots. To execute the programs on these robots, we extended VCC with an additional robot interface that implemented input scene capture and hand actions. Scene capture was achieved through a camera attached to the respective robot's end effector, a gripper. A color image with red, green, and blue channels of the scene was captured by this camera and passed on to the scene parser of the emulator, which created the VCC's initial state. Moving a hand to any location within the workspace and grasping and releasing objects were implemented on robot using a simple Cartesian controller that moved the hand to a given x, y position on a table in a plane at a specified height above the table top. Grasping and releasing an object involved moving the gripper down and closing and opening the gripper. Instead of dragging the object along the table, we moved the object slightly above the table but otherwise respected the same collision constraints as our TW, including the boundary. The interface also mapped any position in VCC's workspace onto x, y coordinates in the robot frame of reference. Program execution took place in the VCC and called scene capture and hand action functions implemented in the robot interface when available. Executing any specific program involved giving the induced concept as a list of primitives with arguments when available and running VCC with robot interface. We used the same visual scene parser for execution on robots as we did for the VCC. We tested six different concepts, including a complex concept that involved executing two concepts in sequence. We used colored foam blocks of different shapes, fruits, and household items as objects and executed programs under different variations of background, number, and shapes and types of objects and with or without distractor objects.

Mapping locations from VCC workspace to robot reference frame requires accurate calibration of camera pose with respect to the robot, and moving robot arm to a specific location requires accurate execution of arm movement. Our UR5 robot had an external RealSense camera attached to the gripper with an accurate calibration. For Baxter, an inbuilt camera inside the gripper was used instead for scene capture. This camera has low resolution and some burnout pixels with an approximate calibration. Because of these limitations, executions were typically accurate and more successful on UR5 compared with Baxter. Movement execution was also faster on UR5 compared with Baxter. Failed runs of program execution were primarily due to grasp failures. These happened more frequently on Baxter, so we tested most of the variations on UR5, which has better movement accuracy.

SUPPLEMENTARY MATERIALS

robotics.sciencemag.org/cgi/content/full/4/26/eaav3150/DC1

Text

Fig. S1. Schematic showing local bounded working memory mappings in VCC for an example program.

Fig. S2. Features extracted from example images used for argument prediction.

Fig. S3. Argument prediction network architecture.

Fig. S4. Examples of valid test input images for three different concepts.

Table S1. List of primitive functions.

Movie S1. The concept of moving yellow objects toward the left and green objects toward the right is taught through schematic images and transferred for execution on robot to separate lemons from limes.

Movie S2. A robot executing the concept of arranging objects in a circle under various settings.

Movie S3. Robots executing the concept of stacking objects on the bottom left in a variety of settings.

Movie S4. Robots executing the concept of moving the yellow object to the bottom left corner and the green object to the top right corner in a variety of settings.

Movie S5. Robots executing the concept of stacking objects vertically in place in a variety of settings.

REFERENCES AND NOTES

1. B. Akgun, M. Cakmak, K. Jiang, A. L. Thomaz, Keyframe-based learning from demonstration. *Int. J. Soc. Robot.* **4**, 343–355 (2012).
2. Y. Duan, M. Andrychowicz, B. C. Stadie, J. Ho, J. Schneider, I. Sutskever, P. Abbeel, W. Zaremba, One-shot imitation learning, in *Advances in Neural Information Processing Systems* (NIPS, 2017), pp. 1087–1098.

3. D.-A. Huang, S. Nair, D. Xu, Y. Zhu, A. Garg, L. Fei-Fei, S. Savarese, J. C. Niebles, Neural task graphs: Generalizing to unseen tasks from a single video demonstration. arXiv:1807.03480 [cs.CV] (10 July 2018).
4. C. Finn, T. Yu, T. Zhang, P. Abbeel, S. Levine, One-shot visual imitation learning via meta-learning. arXiv:1709.04905 [cs.LG] (14 September 2017).
5. H.-Y. F. Tung, A. W. Harley, L.-K. Huang, K. Fragkiadaki, Reward learning from narrated demonstrations. arXiv:1804.10692 [cs.CV] (27 April 2018).
6. L. W. Barsalou, Perceptual symbol systems. *Behav. Brain Sci.* **22**, 577–609 (1999).
7. J. M. Mandler, How to build a baby: II. Conceptual primitives. *Psychol. Rev.* **99**, 587–604 (1992).
8. A. Cangelosi, A. Greco, S. Harnad, *Simulating the Evolution of Language* (Springer, 2002), pp. 191–210.
9. S. Harnad, The symbol grounding problem. *Physica D* **42**, 335–346 (1990).
10. M. Amalric, L. Wang, P. Pica, S. Figueira, M. Sigman, S. Dehaene, The language of geometry: Fast comprehension of geometrical primitives and rules in human adults and preschoolers. *PLOS Comput. Biol.* **13**, e1005273 (2017).
11. J. K. Tsotsos, W. Kruijne, Cognitive programs: Software for attention's executive. *Front. Psychol.* **5**, 1260 (2014).
12. A. M. Turing, On computable numbers, with an application to the entscheidungsproblem. *Proc. Lond. Math. Soc.* **s2-42**, 230–265 (1937).
13. A. Zylberberg, S. Dehaene, P. R. Roelfsema, M. Sigman, The human turing machine: A neural framework for mental programs. *Trends Cogn. Sci.* **15**, 293–300 (2011).
14. J. Von Neumann, *The Computer and the Brain* (Yale Univ. Press, 2012).
15. P. R. Roelfsema, Elemental operations in vision. *Trends Cogn. Sci.* **9**, 226–233 (2005).
16. I. Yildirim, R. A. Jacobs, Learning multisensory representations for auditory-visual transfer of sequence category knowledge: A probabilistic language of thought approach. *Psychon. Bull. Rev.* **22**, 673–686 (2015).
17. M. C. Overlan, R. A. Jacobs, S. T. Piantadosi, Learning abstract visual concepts via probabilistic program induction in a language of thought. *Cognition* **168**, 320–334 (2017).
18. S. Ullman, *High-level Vision: Object Recognition and Visual Cognition* (MIT Press, 1996), vol. 2.
19. J. Johnson, B. Hariharan, L. van der Maaten, J. Hoffman, L. Fei-Fei, C. L. Zitnick, R. Girshick, *Inferring and Executing Programs for Visual Reasoning* (ICCV, 2017), pp. 3008–3017.
20. J. M. Mandler, C. P. Cánovas, On defining image schemas. *Lang. Cogn.* **6**, 510–532 (2014).
21. D. H. Ballard, M. M. Hayhoe, P. K. Pook, R. P. Rao, Deictic codes for the embodiment of cognition. *Behav. Brain Sci.* **20**, 723–742 (1997).
22. P. R. Roelfsema, F. P. de Lange, Early visual cortex as a multiscale cognitive blackboard. *Annu. Rev. Vis. Sci.* **2**, 131–151 (2016).
23. G. Lakoff, R. E. Núñez, Where mathematics comes from: How the embodied mind brings mathematics into being. *AMC* **10**, 12 (2000).
24. D. George, W. Lehrach, K. Kansky, M. Lázaro-Gredilla, C. Laan, B. Marthi, X. Lou, Z. Meng, Y. Liu, H. Wang, A. Lavin, D. S. Phoenix, A generative vision model that trains with high data efficiency and breaks text-based CAPTCHAs. *Science* **358**, eaag2612 (2017).
25. K. Kansky, T. Silver, D. A. Mély, M. Eldawy, M. Lázaro-Gredilla, X. Lou, N. Dorfman, S. Sidor, S. Phoenix, D. George, Schema networks: Zero-shot transfer with a generative causal model of intuitive physics, in *International Conference on Machine Learning* (ICML, 2017), pp. 1809–1818.
26. B. M. Lake, T. D. Ullman, J. B. Tenenbaum, S. J. Gershman, Building machines that learn and think like people, in *Behavioral and Brain Sciences* (2016), pp. 1–101.
27. S. Gulwani, J. Hernández-Orallo, E. Kitzelmann, S. H. Muggleton, U. Schmid, B. Zorn, Inductive programming meets the real world, in *Communications of the ACM* (Association for Computing Machinery, 2015), vol. 58, pp. 90–99.
28. M. Tomasello, Acquiring linguistic constructions, in *Child and Adolescent Development* (2008), p. 263.
29. J. C. Macbeth, D. Gromann, M. M. Hedblom, Image Schemas and Conceptual Dependency Primitives: A Comparison. Technical Report.
30. D. P. Kingma, M. Welling, Auto-encoding variational Bayes. arXiv:1312.6114 [stat. ML] (20 December 2013).
31. S. Gulwani, Dimensions in program synthesis, in *Proceedings of the 12th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP'10)* (ACM, 2010), pp. 13–24.
32. M. Balog, A. L. Gaunt, M. Brockschmidt, S. Nowozin, D. Tarlow, DeepCoder: Learning to write programs. arXiv:1611.01989 [cs.LG] (7 November 2016).
33. X. Chen, C. Liu, D. Song, Towards synthesizing complex programs from input-output examples. arXiv:1706.01284 [cs.LG] (5 June 2017).
34. E. Dechter, J. Malmaud, R. P. Adams, J. B. Tenenbaum, Bootstrap learning via modular concept discovery, in *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)* (IJCAI, 2013), pp. 1302–1309.
35. A. Graves, G. Wayne, I. Danihelka, Neural turing machines. arXiv:1410.5401 [cs.NE] (10 December 2014).
36. D. Lin, E. Dechter, K. Ellis, J. Tenenbaum, S. Muggleton, Bias reformulation for one-shot function induction, in *Proceedings of the 23rd European Conference on Artificial Intelligence* (IOS Press, 2014), pp. 525–530.
37. K. Ellis, L. Morales, M. S. Meyer, A. Solar-Lezama, J. B. Tenenbaum, Dreamcoder: Bootstrapping domain-specific languages for neurally-guided bayesian program learning, in *Neural Abstract Machines and Program Induction Workshop at NIPS 2018* (NIPS, 2018).
38. B. M. Lake, R. Salakhutdinov, J. B. Tnenbaum, Human-level concept learning through probabilistic program induction. *Science* **350**, 1332–1338 (2015).
39. We also tested the effect of increasing the search budget. When increasing it to 3 million programs, 41 concepts are discovered without subroutines and an additional 61 when using subroutines. The effect of subroutines in concept discovery is much more marked in the case of the order-0 model because subroutines are the only mechanism that makes memory available to the model.
40. D. Whitney, E. Rosen, E. Phillips, G. Konidaris, S. Tellex, Comparing robot grasping teleoperation across desktop and virtual reality with ROS reality, in *International Symposium on Robotics Research* (Springer International, 2017), pp. 1–16.
41. Y. Wu, Y. Demiris, Towards one shot learning by imitation for humanoid robots, in *2010 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2010), pp. 2889–2894.
42. D. Hofstadter, M. Mitchell, The Copycat Project: A model of mental fluidity and analogy-making, in *Advances in Connectionist and Neural Computation Theory*, K. J. Holyoak, J. A. Barnden, Eds. (Ablex, 1995).
43. R. M. French, D. Hofstadter, Tabletop: An emergent, stochastic model of analogy-making, in *Proceedings of the 13th Annual Conference of the Cognitive Science Society* (Lawrence Erlbaum Associates, 1991), pp. 175–182.
44. Y. Ganin, T. Kulkarni, I. Babuschkin, S. M. A. Eslami, O. Vinyals, Synthesizing Programs for Images using Reinforced Adversarial Learning. arXiv:1804.01118 [cs.CV] (3 April 2018).
45. J. Tremblay, T. To, A. Molchanov, S. Tyree, J. Kautz, S. Birchfield, Synthetically trained neural networks for learning human-readable plans from real-world demonstrations. arXiv:1805.07054 [cs.RO] (10 July 2018).
46. M. P. Johnson, P. Maes, T. Darrell, Evolving visual routines. *Artif. Life* **1**, 373–389 (1994).
47. A. K. McCallum, Learning visual routines with reinforcement learning, in *AAAI Fall Symposium 1996* (Massachusetts Institute of Technology, 1996), pp. 82–86.
48. I. Horswill, Visual routines and visual search: A real-time implementation and an automata-theoretic analysis, in *International Joint Conference on Artificial Intelligence* (Citeseer, 1995), pp. 56–63.
49. S. Rao, Visual routines and attention, thesis, Massachusetts Institute of Technology (1998).
50. G. Salgian, D. H. Ballard, Visual routines for autonomous driving, *International Conference on Computer Vision* (1998), pp. 876–882.
51. G. Pezzulo, G. Calvi, Toward a perceptual symbol system, in *Proceedings of the Sixth International Conference on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems. Lund University Cognitive Science Studies*, vol. 118.
52. I. Kotseruba, J. K. Tsotsos, STAR-RT: Visual attention for real-time video game playing. arXiv:1711.09464 [cs.CV] (26 November 2017).
53. R. A. Andersen, C. A. Buneo, Sensorimotor integration in posterior parietal cortex. *Adv. Neurol.* **93**, 159–177 (2003).
54. A. Newell, *SOAR: A Cognitive Architecture in Perspective* (Springer, 1992), pp. 25–79.
55. J. M. Lawler, Metaphors we live by. *Language* **59**, 201–207 (1983).
56. O. Kolodny, S. Edelman, The evolution of the capacity for language: The ecological context and adaptive value of a process of cognitive hijacking. *Philos. Trans. R. Soc. Lond. B Biol. Sci.* **373**, 20170052 (2018).
57. N. D. Goodman, J. B. Tenenbaum, T. Gerstenberg, "Concepts in a probabilistic language of thought," (Technical Report 010, Center for Brains, Minds and Machines, 2014).
58. Y. LeCun, Y. Bengio, G. Hinton, Deep learning. *Nature* **521**, 436–444 (2015).
59. J. Schmidhuber, Deep learning in neural networks: An overview. *Neural Netw.* **61**, 85–117 (2015).
60. G. Marcus, Deep learning: A critical appraisal. arXiv:1801.00631 [cs.AI] (2 January 2018).
61. G. Marcus, A. Marblestone, T. Dean, The atoms of neural computation. *Science* **346**, 551–552 (2014).
62. D. L. K. Yamins, J. J. DiCarlo, Using goal-driven deep learning models to understand sensory cortex. *Nat. Neurosci.* **19**, 356–365 (2016).
63. J. K. Tsotsos, *A Computational Perspective on Visual Attention* (MIT Press, 2011).
64. A. Rosenfeld, J. K. Tsotsos, Bridging cognitive programs and machine learning. arXiv:1802.06091 [cs.LG] (16 February 2018).
65. C. von der Malsburg, The what and why of binding: The modeler's perspective. *Neuron* **24**, 95–104 (1999).
66. J. C. Raven, *Raven's Progressive Matrices* (Western Psychological Services, 1938).
67. I. Higgins, N. Sonnerat, L. Matthey, A. Pal, C. P. Burgess, M. Bosnjak, M. Shanahan, M. Botvinick, D. Hassabis, A. Lerchner, Scan: Learning hierarchical compositional visual concepts. arXiv:1707.03389 (2017).
68. S. H. Muggleton, D. Lin, A. Tamaddoni-Nezhad, Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited. *Mach. Learn.* **100**, 49–73 (2015).

69. N. Hay, M. Stark, A. Schlegel, C. Wendelken, D. Park, E. Purdy, T. Silver, D. S. Phoenix, D. George, Behavior is everything: Towards representing concepts with sensorimotor contingencies, in *AAAI Conference on Artificial Intelligence* (AAAI Press, 2018).
70. The starting probabilities are now expressed as transitioning probabilities from *e*. Also, note that, because we know $X_1 = e$ deterministically, $\log p(X_1) = 0$, and therefore, it no longer appears in the expression.

Acknowledgments: We thank the anonymous reviewers for their insightful comments that helped to improve the paper. We thank C. Wendelken, N. Hay, and D. S. Phoenix for reviewing the manuscripts. We thank the Vicarious RCN team for the RCN implementation we used and the Vicarious robotics team for the robot interfaces. **Funding:** This research was funded by Vicarious AI. **Author contributions:** D.G. conceived of the model, implemented the VCC, and designed the experiments. M.L.-G., D.G., and D.L. designed and implemented the program induction algorithms. D.L. and D.G. generated the datasets. D.L. implemented the neural nets, ran the program induction experiments, and generated the plots. J.S.G. implemented the robot interfaces, performed the robot experiments, and generated the videos. D.G., J.S.G., and D.L. generated the figures. D.G. and M.L.-G. wrote the paper with the assistance of D.L. and J.S.G. **Competing interests:** Vicarious AI has filed a U.S. Patent Office application (number 62/727,162) related to this work. The authors declare that they have no competing financial interests. **Data and materials availability:** Program induction code and datasets used in the experiments will be made available at www.vicarious.com. All other data needed to evaluate the conclusions in the paper are present in the paper and/or the Supplementary Materials.

Submitted 5 September 2018
Accepted 19 November 2018
Published 16 January 2019
10.1126/scirobotics.aav3150

Citation: M. Lázaro-Gredilla, D. Lin, J. S. Guntupalli, D. George, Beyond imitation: Zero-shot task transfer on robots by learning concepts as cognitive programs. *Sci. Robot.* **4**, eaav3150 (2019).

Science Robotics

Beyond imitation: Zero-shot task transfer on robots by learning concepts as cognitive programs

Miguel Lázaro-Gredilla, Dianhuan Lin, J. Swaroop Guntupalli, and Dileep George

Sci. Robot. **4** (26), eaav3150. DOI: 10.1126/scirobotics.aav3150

Science Robotics (ISSN 2470-9476) is published by the American Association for the Advancement of Science. 1200 New York Avenue NW, Washington, DC 20005. The title *Science Robotics* is a registered trademark of AAAS.