Powering up causal generalization: A model of human conceptual bootstrapping with adaptor grammars

Anonymous CogSci submission

Abstract

Human learning and generalization benefit from *bootstrapping*: we arrive at complex concepts by starting small and building upon past successes. In this paper, we examine a computational account of causal conceptual bootstrapping, and describe a novel experiment in which the sequence of training data results in a dramatic order effect: participants succeed in identifying a compound concept only after experiencing training data in a "helpful" order. Our computational model represents causal relations as reusable, modular programs, which can themselves be "chunked" and flexibly reused to tackle more complex tasks. Our specific approach is based in combinatory logic and adaptor grammars, building on previous theories that posit a "language of thought" for concept representation, but making the learning process more sensitive to a learner's experiences than any particular choice of conceptual primitives. Crucially, we demonstrate that a caching mechanism like that used in adaptor grammars is key to explaining human-like bootstrapping patterns in causal generalization.

Keywords: Causal reasoning; generalization; bootstrapping; adaptor grammar; approximate Bayesian inference

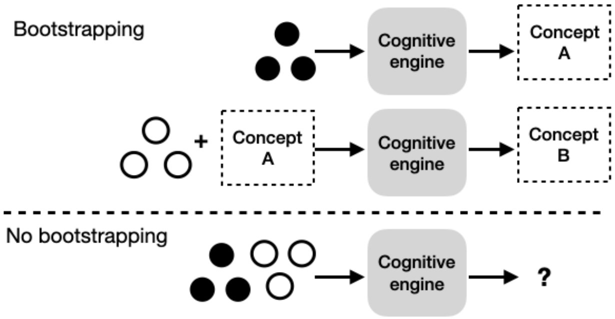

Figure 1: Bootstrapping forms complex concepts by extending existing concepts to account for novel observations. Without the process of bootstrapping, the world may be too complex to conceptualize. Dots represent data points.

Introduction

Human babies do not stop at recognizing "one", "two" or "lots" of something. Soon enough, most "bootstrap" their way to a conceptual understanding of a number system built on the successor relationship (Carey, 2004; Piantadosi et al., 2012). This unlocks more complex mathematical concepts, paves the way to novel mathematical discoveries, and ultimately technological feats like sending rockets into space. While many previous computational approaches to generalization suggested that people are equipped with rich learned representations in novel situations (e.g. Kemp et al., 2012; Wu et al., 2018), bootstrapping puts forward a more constructive and compositional view: We don't just compute a posterior predictive distribution given our experience, but also build on existing knowledge, enrich it with insights from new observations, and that may lead to the generation of new concepts (Figure 1). When the world is too complex to make sense of wholesale, bootstrapping provides a way to start small and build incrementally on past successes in order to ultimately arrive at richer representations that can unlock useful new options (Carey, 2004; Khan et al., 2011; Krueger & Dayan, 2009; Piantadosi et al., 2012). Such a process can also explain the nested structure of our concepts, reflected in the high degree of compositionality and transferability exhibited in human learning (Lake et al., 2017).

This dynamic and adaptive view of generalization calls for a systematic mental mechanism that enables efficient concept formation from re-combination of existing concepts. Under this view, we expect learning and generalization to be influenced by the order in which learners see evidence: an order from simpler problems to ones that reply on more complex or numerous concepts should support faster and better learning. This idea has inspired earlier work under the label of "curriculum learning" in the machine learning literature (Bengio et al., 2009; Graves et al., 2017; Mao et al., 2019). However, human conceptual representations are more than weight-based search over an error space, and recent research has overcome some of these limitations using compositional and symbolic frameworks for abductive generalization, drawing upon a range of methods including Probabilistic Context-Free Grammars (PCFGs, Bramley et al., 2018; Goodman et al., 2008; Zhao et al., 2021), lambda calculus (Ellis et al., 2021; Piantadosi et al., 2016), fragment grammars (O'Donnell et al., 2009), etc.

Inspired by Liang et al. (2010) and Dechter et al. (2013), in this paper we propose a computational account of human-like causal concept bootstrapping based on combinatory logic (CL, Schönfinkel, 1924) and adaptor grammars (AGs, Johnson et al., 2007). As a Bayesian-symbolic model, our formalization shares all of the virtues of the PCFG framework, but crucially, supports abstraction and reuse in ways that a PCFG framework does not. We describe a causal learning experiment, showing our model predicts bootstrapping order effects—benefiting from a "facilitatory" curriculum order while suffering under a misleading "learning trap" curriculum (cf. Gelpi et al., 2020). We show that approaches based on exhaustive inference over a concept grammar (without bootstrapping), fail to capture these patterns.

Modeling Concept Bootstrapping

We adapted our task interface from Zhao et al. (2021): In each learning trial, participants see an agent object collide with a stationary recipient object, which consequently transforms in some way. Participants are instructed to reason about the causal relationship between features of the agent and recipient objects, and the resulting changes in the recipient.

Causal relationships as functional programs

We treat causal representations as fundamental cognitive representations (cf. Chater & Oaksford, 2013) for predicting, explaining and controlling the world (Gopnik et al., 2007; Griffiths et al., 2010; Sloman, 2005). In particular, we use CL to formalize causal concepts as programs that take the agent and recipient objects as input, and output the result object. We assume that object features are available and salient to participants, and these features could change in a result object as functions[1], making our account compatible with a structural equation model perspective (Duncan, 1975; Pearl, 2000).

Functional terms CL programs are composed of *terms* and input *variables*. Terms are interpreted as functions by definition, and can be composed iteratively to generate new terms. Starting with our assumption that relevant features are salient to the learner, we let function $getFeature(o) = v$ take an object o as input and return its feature value v; function $setFeature(o,v) = o'$ sets object o's feature value to v, returning an updated object o'. For our task, as illustrated in Figure 2a, we consider a minimal set of base terms: $getSpot()$, $getStripe()$, $getSegment()$, and $setSegment()$. Since numbers of spots, stripes or segments are all numerical, we include some operations over these feature values as additional base terms (or chunks that are salient from past experience): addition $add(v,u) = v + u$, subtraction $sub(v,u) = v - u$, and multiplication $mult(v,u) = v \times u$,

Types Since terms are functions, they are naturally constrained by their input domains and output co-domains, known as being "typed". Taking object (obj) and numeric value (num) as base types, a type t for a term is written as $t_I \rightarrow t_O$, where t_I and t_O are types for the input and output respectively. Table 1 lists the type signatures for the primitive terms we introduced earlier. Conventionally, type signatures are written as subscripts, like $getSpot_{obj \rightarrow num}$. Type signatures are critical for ensuring valid compositions. For instance, we can plug in any subprogram the returns a number as one argument for *add*. However, we cannot use *setSegment* as an argument for *add*, because *setSegment* returns an object while *add* requires numbers as inputs.

Table 1: Base terms.

Terms	Type signature
getSpot, getStripe, getSegment	obj → num
add, sub, mult	num → num → num
setSegment	obj → num → obj

Routing variables When evaluating nested functions, it is essential to make sure input variables are sent to the right place. In lambda calculus, for example, this is done by ensuring the unique and uniform use of symbols representing the same variable throughout the nested layers. As a result, when composing subprograms, additional machinery is necessary to determine what symbols are re-used and where.

To solve this variable binding problem, CL introduces some terms that serve as "routers" (Figure 2b): For a tree-like structure $[router, x, y]$, router **B** routes an incoming variable z to the right of the tree—z is first fed to the right-hand y, and the result of this is then sent to x. Similarly, router **C** routes z to the left, router **S** sends z to both sides, and router **I** is an identity function that returns an input as it is. For N input variables, we can concatenate N routers in corresponding order.

Causal programs With variables, terms, types and routers all at hand, we are now all set to consider an example program as unpacked in Figure 2c:

[**CS** [**BB** *setSegment* [**BC** [**B** *sub getSegment*] *getSpot*]] **I**]

Evaluating this example program with the agent and recipient objects in Figure 2a as input, the first router **CS** routes the agent object to the left (solid arrows), and routes recipient object to both sides (dotted arrows), so on and so forth. After instantiating all the variables, this program reads as: "take the recipient and make its number of segments to be its original number of segments minus agent's number of spots", and outputs a result object with two segments.

Program evaluation Let observational data $D = \langle X, Y \rangle$ where X are input data and Y the output. The likelihood function of program m_t producing observational data D is given by:

$$P(m_t|D) = \begin{cases} 1 & \text{if } m_t(X) = Y \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

Taking the agent and recipient objects in Figure 2a as input, the example program in Figure 2c returns the result object that is a stick of two segment, hence its likelihood for producing the example task is 1.

Bootstrapping with adaptor grammars

The core difference between AGs and PCFGs is that AGs allow caching: a generated program can be added to the library of

[1] While these assumptions are primitive in our model, we could also assume other primitives that can be composed to achieve the sames ends along with earlier experience that leads to their composition, e.g., being asked, as a child might, to count the spots on an object.

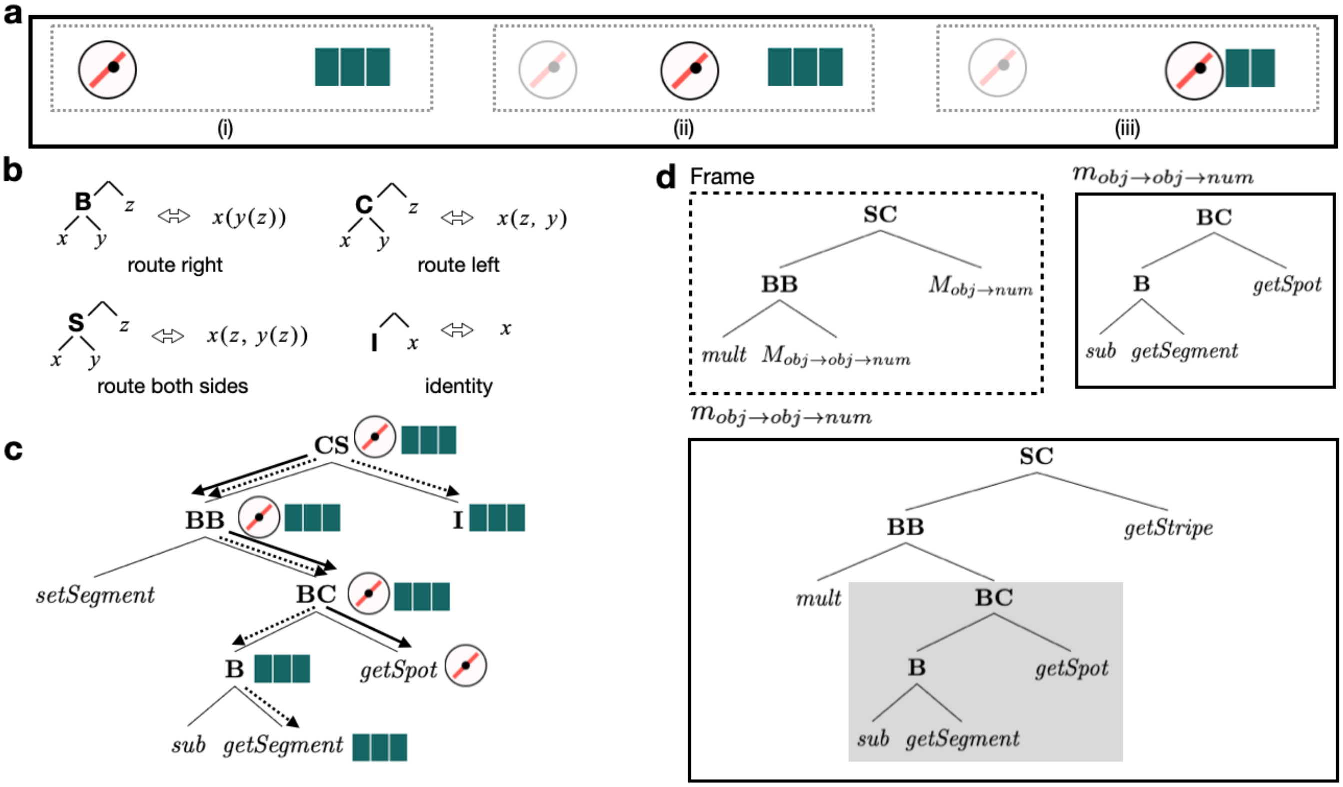

Figure 2: a. A task example: (i) A magic egg (agent) with one stripe and one spot (ii) moves rightward and hits a stick (recipient) of three segments, resulting in (iii) the stick becomes one-segment shorter (result). Translucent icons mark the starting position of the magic egg, but in the experiment movement was animated. b. Visualization for CL routers, adapted from Liang et al. (2010). c. Example program. Black arrows denote routing of the agent (magic egg), and dotted arrows for routing of the recipient (stick). d. Example frame (dotted box) and example programs. Shaded area in the bottom program reuses the program on top-right.

"primitives" for later reuse; program generation can result from either composing a new program, or sampling directly from the cache. We may think about these cached programs as "concepts": They possess some internal complexity, serve certain functional aims, and more importantly, can be reused directly without having to be "rediscovered" by regenerating all the internal parts again. The caching mechanism of AGs thus facilitates bootstrapping via chunking useful subprograms and reusing them as building blocks anywhere that their type constraints allow (Liang et al., 2010).

Generative process As in PCFGs, AGs implicitly define a distribution over programs via a generative process. Let $\mathcal{L}$ be a program library consisting of base terms and/or some programs, with probability λ_1 grammar $\mathcal{G}$ contructs new programs of type t, and otherwise it returns a cached program of type t with probability λ_2. We employ a tail recursion for the construction step as in Dechter et al. (2013) in order to efficiently satisfy type constraints in Table 1. That is, we start by sampling a left-hand side term *LHS* whose output type is the same as the output type of t. Based on how many variables are fed to this stage, grammar $\mathcal{G}$ then samples a router *RT* of corresponding length that sends these variables to either/both branches. Since both *LHS* and router *RT* are given, now the type signature for the right-hand side of the tree is fully specified, because it has all the input types (routed by *RT*) and a required output type (to feed into *LHS*). Therefore, we apply the same procedure iteratively to get this right-hand side subprogram *RHS*, returning the final program [*RT LHS RHS*]. The constructed program [*RT LHS RHS*] is then added to the program library $\mathcal{L}$ (caching).

Each step in this generative process comes with a probability distribution. For the starting program library $\mathcal{L}$, we assume a uniform distribution over terms that share the same type signature. We also assume a uniform distribution over routers sharing the same number of variables to route. Following the notation in Liang et al. (2010), for a collection of terms C_t of type signature t, let N_t be the number of distinct elements in C_t, and M_z the number of times z occurs in C_{z_t}:

$$\lambda_1 = \frac{\alpha_0 + N_t d}{\alpha_0 + |C_t|}, \quad \lambda_2 = \frac{M_z - d}{|C_t| - N_t d}. \tag{2}$$

Hyper-parameters $\alpha_0 > 0$ and $0 < d < 1$ control the amount of sharing and reuse. Since λ_1 is proportional to $\alpha_0 + N_t d$, the smaller α_0 and d are, the less construction and more sharing we have. Similarly, λ_2 is proportional to M_z, hence the more

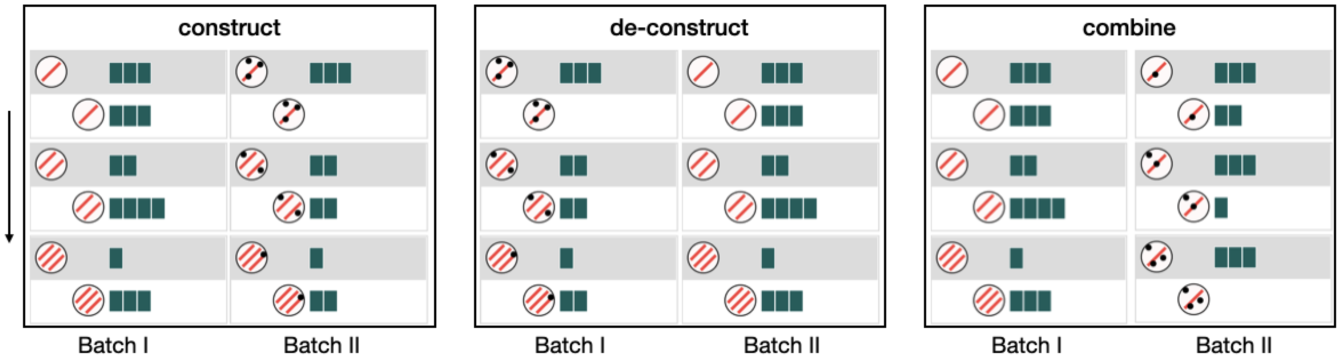

Figure 3: Experiment stimuli.

frequently a program is cached, the higher weight it gets, regardless of its internal complexity.

Approximate Bayesian inference Given this probabilistic model, we are faced with the challenge of efficiently approximating a posterior distribution over latent programs given learning data, according to the prior distribution (Equations 2) and likelihood function (Equation 1). Following previous work suggesting that human learners make inferences by sampling from an approximate posterior instead of tracking the entire posterior space of possibilities (Bramley et al., 2017), we use known methods for sampling from Pitman-Yor processes (Pitman & Yor, 1997), such that conditional on a program library at any given moment, learners can make appropriate inferences about the probabilities of different explanations for new or salient events. Concretely, we use a Gibbs sampler for program library L: for the i-th iteration, conditional on the library from previous iteration L_{i-1}, sample an updated library L_i and add it to the collection of samples. For the sampling step, let library L_{i-1} generate programs with probabilities defined above and calculate their likelihoods with respect to learning data D. The caching mechanism of AG will add consistent programs into library L_{i-1}, or increase the counter for those already present in L_{i-1}, resulting in an updated library L_i.

In practice, our learning data is very sparse, hence we adopt both breadth-first search (Knuth, 1973) and beam search (Hayes-Roth et al., 1977) to facilitate search for programs that can produce learning data. For the outer loop, we use "frames" for intermediate programs built with typed placeholders (Figure 2d). Fixing a generation depth, we first enumerate a set of frames $\mathcal{F}$. Next, sample a frame from $\mathcal{F}$ according to generation probabilities. The sampled frame can then be unfolded, replacing its placeholders with programs of required types, yielding a set of fully-articulated programs M (Figure 2d). If some programs $M^* \subseteq M$ produce learning data with likelihood 1, we stop the search; otherwise, we sample another frame from $\mathcal{F}$ and repeat. If no programs are consistent with data after depleting frame set $\mathcal{F}$, we increase depth by 1 and repeat until a maximal cap is met. Because of this comprehensive search-check-sample procedure, we expect our Gibbs sampler to approximate the true posterior quickly and without the need for extensive burn-in.

Generalization predictions We can run the generative procedure of grammar $\mathcal{G}$ using the sampled libraries 10,000 times to approximates a distribution $Dist_M$ over latent causal programs, and make generalization predictions about new partially observed data $D^* = \langle X^*, ? \rangle$, producing a predicted distribution $Dist_P$ over generalizations.

Experiment

We evaluated our model in a two-phase causal learning task. The experiment was preregistered in OSF registry.

Methods

Participants 165 participants ($M_{\text{age}} = 31.8 \pm 9.9$) were recruited from Prolific Academic. Participants received a base payment of £1.25 and performance-based bonuses (highest paid £1.93). The task took 14.2 ± 6.1 minutes. No participant was excluded from analysis.

Design We used a task animation as illustrated in Figure 2a. The agent object A was visualized as a circle that moved in from the left of screen and collided with the recipient R. The agent object A varied in its number of stripes and (randomly positioned) spots. The recipient object R took the form of a stick made up of a number of cube shaped segments. During learning, all feature values were between 0 and 3. For generalization tasks, an arbitrary segment number (up to 16) could be selected. The true rule determining the recipient's number of segments was $\texttt{Segments}(R') \leftarrow \texttt{Stripes}(A) \times \texttt{Segments}(R) - \texttt{Spots}(A)$.

We examined three between-subject learning conditions we call *construct*, *de-construct*, and *combine* (Figure 3). Each condition contains six learning trials—pairs of agent and recipient objects. These six pairs are divided into two batches, I and II. For the *construct* condition, in Batch I learners only see examples that vary in terms of the stripe feature (with zero dots in each case). Then, in Batch II, they see examples with varied spot features on top of the stripe features. The *de-construct* condition contains the same six trials but swaps batch I and batch II such that the first three trials have both spots and stripes varying across them, and the second batch only varies

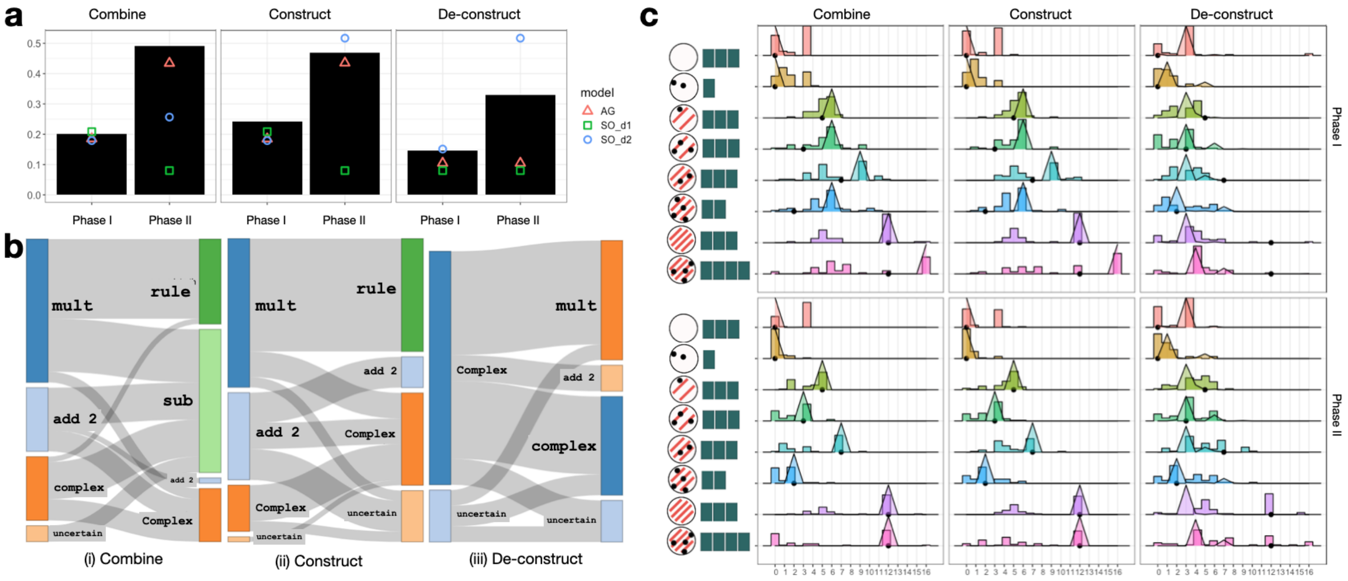

Figure 4: Experiment results per condition and per task phase. a. Generalization prediction accuracy. Bars are participants data, colored shapes mark accuracy per model based on fitted model predictions. b. Labeled self-reports. Coding scheme and full dataset is available at OSF. c. Generalization predictions per task (rows) by participants (bars) and the best fitting AG model (densities). For each panel, x-axis is result object's number of segments, ranging from 0 to 16.

stripes. For the *combine* condition, batch I is identical to batch I in the construct condition. However, in *combine* batch II, the spots are varied while the number of stripes is held constant at 1. In sum, conditions *construct* and *de-construct* present the identical learning evidence but in a different order. Condition *combine* presents evidence of comparable diagnosticity to the other two conditions but leaves the combination of roles of the stripes and the spots ambiguous to the learner.

According to our computational model, participants in the *combine* and *construct* conditions should be able to learn a multiplication relationship with information in batch I, and then build upon this knowledge to conclude the compound ground truth relationship with information in batch II. Participants in the *de-construct* condition, however, should fail to do so for the lack of facilitatory learning curriculum. We measured generalization performance using both free-text self-reports and eight forced-choice tasks on novel pairs of objects, selected by maximally differentiating between salient alternative rules (according to our grammar) and covering edge cases like a zero-spot & zero-stripe agent.

Procedure Each participant was randomly assigned to one of the three learning conditions. After reading instructions and passing a comprehension quiz, they went through experiment phase I and then phase II. In each phase, a participant tested three learning examples from the corresponding batch [2] by clicking a "Test" button and observing the animated outcome, and then were asked to write down their guesses about the underlying causal relationships, and made generalization predictions for eight pairs of novel objects. Once tested, a visual summary of the learning example including the initial and final state of the recipient was added to the screen and remained visible until the end of the experiment. Generalization trials appeared sequentially. Once a prediction was made the trial was replaced by the next one. The pairs of generalization objects in both phase I and phase II are the same, but their presentation orders were randomized.

Results

Bootstrapping effects We found evidence for causal concept bootstrapping from both participants' generalization accuracy (Figure 4a) and their self-reports on causal relationships (Figure 4b). For Phase II generalization predictions, participants in the *combine* and *construct* conditions achieved average accuracies of 49.1% and 46.9%, while participants in the *de-construct* condition of only 33.0% (chance is 5.9%). A repeated measures ANOVA predicting each participant's taskwise generalization accuracy (165 × 8) with condition as between-subject factor and phase as within-subject repeated measure confirmed main effects of condition ($F(2,1317) = 15.26, p < .001$) and phase ($F(1,1317) = 239.91, p < .001$), and an interaction between condition and phase ($F(2,1317) = 4.06, p = .018$). Pairwise comparison of conditions revealed a significant difference between *construct* and *de-construct*, $t(1317) = -4.98, p < .001, 95\%\text{CI}[-0.35, -0.12]$, between *combine* and *de-construct*, $t(1317) = -4.56, p < .001, 95\%\text{CI}[-0.33, -0.10]$, but not between *combine* and *construct*, $t(1317) = -0.37, p = 1$. Recall that at the end of Phase II, participants in the *construct* and *de-construct* conditions have observed identical learning information, only in different orders. These results demonstrate the bootstrapping

[2] We use "batch" for stimuli and "phase" for experimental stages.

effect—composing the right subprogram contributes to successful generalization in subsequent, more complex scenarios.

From self-reports, we found that while 39.3% of participants in the *construct* condition and 29.6% in the *combine* condition reported causal relationships as the intended ground truth rule, no one in the *de-construct* condition did so (Figure 4b), $F(2,) = 10.79, p < 0.001$. A deeper dive into those self-reports revealed that, for those participants who guessed the $\texttt{Stripes}(A) \times \texttt{Segments}(R)$ subprogram in Phase I, 75.9% of them in the *construct* condition and 55.6% in the *combine* condition landed on the correct ground truth rule in Phase II. This directly supports our concept reuse model. For participants in the *de-construct* condition, 81.8% came up with complex rules in Phase I, that may draw upon position of spots, relationships between the number of spots and stripes, etc. In Phase II, only 41.8% of participants in the *de-construct* condition reported the $\texttt{Stripes}(A) \times \texttt{Segments}(R)$ subprogram, fewer than the 51.8% of *construct* and 50% of *combine* condition in Phase I, indicating a garden-path effect (cf. Gelpi et al., 2020) that people might get lost when wandering into a forest of complicated ideas.

Model fits We compare four models: a random selection model as baseline, the adaptor grammar model as introduced above, and two grammar-based models that, like "rational rules" models (Goodman et al., 2008; Piantadosi et al., 2016), omit the concept caching and re-use that distinguishes ours and leads to the distinctive order effects we predict. We call these grammar-based models *search-only*: they use standard Bayesian updating and a prior distribution over an enumerated set of causal programs generated with fixed depth $d = 1$ and $d = 2$ using the same causal functional program setup laid out in the modeling section, and the same likelihood function as in Equation 1. Generalization predictions are computed via marginalization over posterior distribution of causal programs given each task. To account for noise in predictions, we fit a softmax function with a temperature parameter τ on the three computational models (Luce, 1959). Let $P(r'|d)$ be the posterior predictive distribution over candidate length of segments in generalization tasks:

$$P(\text{choice}) = \frac{e^{P(r'|d)\cdot(1/\tau)}}{\sum_{x\in r'} e^{P(x|d)\cdot(1/\tau)}}. \tag{3}$$

Table 2 summarizes model fitting results. All three computational models perform much better than random baseline, demonstrating the power of causal programs in capturing human intuitive causal reasoning. Moreover, adaptor grammar model outperforms the other two search-only models and fits best overall.

Figure 4a illustrates generalization accuracy of each (fitted) computational model. The adaptor grammar model is the only one that can capture the increase in generalization accuracy for the *construct* and *combine* conditions, along with low accuracy in the *de-construct* condition. The search-only ($d = 1$) model cannot bootstrap in Phase II in the *construct* and *combine* conditions because it has no mechanism of reuse. The search-only ($d = 2$) model shows a weak bootstrapping effect in the *combine* condition because it assigns equal preference for alternative compatible causal programs and does not favor reuse. It also achieves an overly high accuracy in Phase II in the *de-construct* condition due to its deeper search depth that allows it to generate the ground truth rule directly.

Table 2: Model Fitting Results.

Model	τ	Log likelihood	BIC
Baseline	-	-7319.63	14639
Search-only ($d = 1$)	2.79	-6738.77	13485
Search-only ($d = 2$)	2.84	-6285.46	12579
Adaptor Grammar	2.52	**-6168.96**	**12348**

Figure 4c plots participants' generalizations with model AG's predictions, revealing the close alignment between the two in addition to BIC and accuracy measures.

General Discussion

We proposed a computational model based on combinatory logic and adaptor grammars as an account for human conceptual bootstrapping, and grounded it in an object-based causal generalization task. Our formalization synthesizes aspects of human intelligence that have long eluded fixed-form symbolic and subsymbolic accounts (Valentin et al., 2021). In particular, our use of adaptor grammars rather than PCFGs—common to other recent treatments of constructivist inference (Bramley et al., 2021; Goodman et al., 2008)—allowed us to capture flexible type-constrained reuse, and so reproduce learning order effects exhibited by participants. That is, our framework explains both how participants succeed in our *construct* and *combine* conditions but also why they failed in the *de-construct* condition.

These results also demonstrate the importance of curriculum design. It was striking that a simple manipulation of the order in which six examples were shown made the difference between 39.3% of people identifying how a causal system works and 0% doing so. Going beyond neural-network based work on curriculum learning (Bengio et al., 2009; Khan et al., 2011), our account sheds lights on how to design effective curricula to build up to complex concepts. We show it is critical to teach concepts early that can later be reused to aid in grasping more complex concepts (Dechter et al., 2013; Krueger & Dayan, 2009).

In sum, our model provides a mechanistic account of chunking and reuse in higher level cognition (Carey, 2004; Gobet et al., 2001; Klein, 2017), and highlights how these processes produce the patterns of flexibility, efficiency, and compositionality that are hallmarks of human cognition.

References

Bengio, Y., Louradour, J., Collobert, R., & Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning* (pp. 41–48).

Bramley, N. R., Dayan, P., Griffiths, T. L., & Lagnado, D. A. (2017). Formalizing Neurath's ship: Approximate algorithms for online causal learning. *Psychological Review*, *124*(3), 301.

Bramley, N. R., Heuser, G., & Xu, F. (2021). Computational constructivism: Developmental differences in active inductive inference. *Under review*.

Bramley, N. R., Rothe, A., Tenenbaum, J., Xu, F., & Gureckis, T. (2018). Grounding compositional hypothesis generation in specific instances. In *Proceedings of the 40th annual meeting of the cognitive science society.*

Carey, S. (2004). Bootstrapping & the origin of concepts. *Daedalus*, *133*(1), 59–68.

Chater, N., & Oaksford, M. (2013). Programs as causal models: Speculations on mental programs and mental representation. *Cognitive Science*, *37*(6), 1171–1191.

Dechter, E., Malmaud, J., Adams, R. P., & Tenenbaum, J. B. (2013). Bootstrap learning via modular concept discovery. In *Twenty-third international joint conference on artificial intelligence.*

Duncan, O. D. (1975). Introduction to structural equation models.

Ellis, K., Wong, C., Nye, M., Sablé-Meyer, M., Morales, L., Hewitt, L., ... Tenenbaum, J. B. (2021). Dreamcoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In *Proceedings of the 42nd acm sigplan international conference on programming language design and implementation* (pp. 835–850).

Gelpi, R., Prystawski, B., Lucas, C. G., & Buchsbaum, D. (2020). Incremental hypothesis revision in causal reasoning across development. In *Proceedings of the 42th annual conference of the cognitive science society.*

Gobet, F., Lane, P. C., Croker, S., Cheng, P. C., Jones, G., Oliver, I., & Pine, J. M. (2001). Chunking mechanisms in human learning. *Trends in cognitive sciences*, *5*(6), 236–243.

Goodman, N. D., Tenenbaum, J. B., Feldman, J., & Griffiths, T. L. (2008). A rational analysis of rule-based concept learning. *Cognitive Science*, *32*(1), 108–154.

Gopnik, A., Schulz, L., & Schulz, L. E. (2007). *Causal learning: Psychology, philosophy, and computation*. Oxford University Press.

Graves, A., Bellemare, M. G., Menick, J., Munos, R., & Kavukcuoglu, K. (2017). Automated curriculum learning for neural networks. In *Proceedings of the 34th international conference on machine learning* (pp. 1311–1320).

Griffiths, T. L., Chater, N., Kemp, C., Perfors, A., & Tenenbaum, J. B. (2010). Probabilistic models of cognition: Exploring representations and inductive biases. *Trends in cognitive sciences*, *14*(8), 357–364.

Hayes-Roth, P., Fox, M., Gill, G., DJ, M., & Reddy, R. (1977). Speech understanding systems: Summary of results of the five-year research effort. *Department of Computer Science*.

Johnson, M., Griffiths, T. L., Goldwater, S., et al. (2007). Adaptor grammars: A framework for specifying compositional nonparametric bayesian models. *Advances in neural information processing systems*, *19*, 641.

Kemp, C., Shafto, P., & Tenenbaum, J. B. (2012). An integrated account of generalization across objects and features. *Cognitive Psychology*, *64*(1-2), 35–73.

Khan, F., Mutlu, B., & Zhu, J. (2011). How do humans teach: On curriculum learning and teaching dimension. In *Advances in neural information processing systems* (pp. 1449–1457).

Klein, G. A. (2017). *Sources of power: How people make decisions*. MIT press.

Knuth, D. E. (1973). Sorting and searching. CUMINCAD.

Krueger, K. A., & Dayan, P. (2009). Flexible shaping: How learning in small steps helps. *Cognition*, *110*(3), 380–394.

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., & Gershman, S. J. (2017). Building machines that learn and think like people. *Behavioral and Brain Sciences*, *40*.

Liang, P., Jordan, M. I., & Klein, D. (2010). Learning programs: A hierarchical Bayesian approach. In *Proceedings of the 27th international conference on machine learning (icml-10)* (pp. 639–646).

Luce, R. D. (1959). *Individual choice behavior*. Wiley.

Mao, J., Gan, C., Kohli, P., Tenenbaum, J. B., & Wu, J. (2019). The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. *International Conference on Learning Representations*.

O'Donnell, T. J., Tenenbaum, J. B., & Goodman, N. D. (2009). Fragment grammars: Exploring computation and reuse in language.

Pearl, J. (2000). *Causality: Model, reasoning, and inference*. Cambridge University Press.

Piantadosi, S. T., Tenenbaum, J. B., & Goodman, N. D. (2012). Bootstrapping in a language of thought: A formal model of numerical concept learning. *Cognition*, *123*(2), 199–217.

Piantadosi, S. T., Tenenbaum, J. B., & Goodman, N. D. (2016). The logical primitives of thought: Empirical foundations for compositional cognitive models. *Psychological Review*, *123*(4), 392.

Pitman, J., & Yor, M. (1997). The two-parameter poisson-dirichlet distribution derived from a stable subordinator. *Annals of Probability*, *25*, 855–900.

Schönfinkel, M. (1924). Über die bausteine der mathematischen logik. *Mathematische Annalen*(92), 305–316.

Sloman, S. A. (2005). *Causal models: How people think about the world and its alternatives*. Oxford University Press.

Valentin, S., Zhao, B., Jiang, C., Bramley, N. R., & Lucas, C. (2021). Symbolic and sub-symbolic systems in people and machines. In *Proceedings of the 43th annual meeting of the cognitive science society.*

Wu, C. M., Schulz, E., Speekenbrink, M., Nelson, J. D., & Meder, B. (2018). Generalization guides human exploration in vast decision spaces. *Nature Human Behaviour*, *2*(12), 915–924.

Zhao, B., Lucas, C. G., & Bramley, N. R. (2021). How do people generalize causal relations over objects? a nonparametric bayesian account. *Computational Brain & Behavior*.