The computational origin of representation and conceptual change

Steven T. Piantadosi
University of Rochester

Abstract

Each of our theories of mental representation provides some insight into how the mind works. However, these insights often seem incompatible, as the debates between symbolic, dynamical, emergentist, sub-symbolic, grounded, and rational approaches to cognition attest. Mental representations—whatever they are—must share many features with each of our theories of representation, and yet there are few hypotheses about how a synthesis could be possible. Here, I develop a theory of the underpinnings of symbolic cognition that shows how sub-symbolic dynamics may give rise to higher-level cognitive representations of structures, systems of knowledge, and algorithmic processes. This theory implements a version of *conceptual role semantics* by positing an internal universal language for isomorphism in which learners may create mental models with arbitrary dynamics. The theory formalizes one account of how novel conceptual content may arise, allowing us to explain how even elementary logical and computational operations may be learned. I provide an implementation that learns to represent a variety of structures, including logic, number, kinship trees, regular languages, context-free languages, domains of theories like magnetism, dominance hierarchies, list structures, quantification, and computational primitives like repetition, reversal, and recursion. The account is based on simple discrete dynamical processes that could be implemented in a variety of different physical or biological systems. In particular, I describe how the required dynamics can be directly implemented in an existing connectionist framework. The resulting theory provides an "assembly language" for cognition, where high-level theories of representation, computation, and conceptual change can be translated into simple underlying dynamics.

Keywords: language of thought; conceptual role semantics; conceptual change; combinatory logic

Introduction

At the core of cognitive science is an embarrassing truth: we do not know what mental representations are like. Many ideas have been developed, from the field's origins in symbolic AI (Newell & Simon, 1976), to parallel and distributed representations of connectionism (Rumelhart & McClelland, 1986; Smolensky & Legendre, 2006), embodied theories that emphasize the grounded nature of the mental (Barsalou, 2008, 2010), Bayesian accounts of structure learning and inference (Griffiths, Chater, Kemp, Perfors, & Tenenbaum, 2010; Tenenbaum, Kemp, Griffiths, & Goodman, 2011), theories of cognitive architecture (Newell, 1994; Anderson, Matessa, & Lebiere, 1997; Anderson et al., 2004), and those based on mental models, simulation (Craik, 1967; Johnson-Laird, 1983; Battaglia, Hamrick, & Tenenbaum, 2013), or analogy (Gentner & Stevens, 1983). These research programs have developed in concert with foundational debates in the philosophy of mind about what kinds of things concepts may be (Margolis & Laurence, 1999), with similarly diverse and seemingly incompatible answers.

This paper develops a cognitive framework that attempts to unify a variety of ideas about how mental representations may work. I argue that what is needed is an *intermediate* bridging theory that lives below the level of symbols and above the level of neurons or neural network nodes. The formalism I present shows how it is possible to implement high-level symbolic constructs permitting arbitrary (Turing-complete) computation in a system that is simple, parallelizable, and addresses foundational questions about meaning. Although the particular instantiation I describe is an extreme simplification, its general principles, I'll argue, are likely to be close to the truth.

The theory I describe is a little unusual in that it is built almost exclusively out ideas that have been independently developed in multiple subfields of cognitive science. The logical formalism comes from mathematical logic and computer science in the early 1900s. The philosophical and conceptual framework has been well-articulated in prior debates. The inferential theory builds on work in theoretical AI and Bayesian cognitive science. The overall goal of finding symbolic cognition below the level of symbols comes from connectionism and other sub-symbolic accounts. An emphasis on the importance of getting symbols eventually comes from the remarkable properties of human language, as well as centuries of thought in philosophy and mathematics about what kinds of formal systems may capture thought. What is new is the unification of these ideas into a framework that can be shown to learn complex symbolic processes and representations that are grounded in simple underlying dynamics without dodging key questions about meaning and representation.

The resulting theory formulates hypotheses about how fundamental *conceptual change* can occur, both computationally and philosophically. A pressing mystery is how children might start with their initial knowledge and come to develop rich and sophisticated systems of representation. The problem is deepest when we consider simple logical capacities—for instance, the ability to represent boolean values, compute syllogisms, follow logical/deductive rules, use number, or process conditionals and quantifiers. If infants do not have some of these abilities, we are in need of a learning theory that can explain where such computational processes might come from. Yet, it is hard to imagine how a computational system that does not know these could function. From what starting point could learners possibly construct Boolean logic, for instance? Is it possible to compute and *not* know logic or number? The answer provided in this paper is emphatically *yes*—computation is possible without any explicit form of these operations, and learning systems can be made that construct and test these logical systems as hypotheses. Conceptual change and representational learning is possible, from the very basics of computation to our most sophisticated logical and algorithmic representations. The goal of this paper is describe one way it is possible.

The paper is organized as follows: In the next section, I describe a leading example of symbolic cognitive science, Fodor's *Language of Thought* (LOT) theory (Fodor, 1975, 2008). The LOT motivates the need for structured, compositional representations, but leaves at its core unanswered questions about how the symbols in these representations come to have meaning. I then discuss *conceptual role semantics* (CRS) as an account of how meaningful symbols may arise. The problem with CRS is that it has no implementations, leaving its actual mechanics vague and unspecified. To develop a formal version of CRS, I describe *combinatory logic*, and a key operation in it, *Church encoding*, which permits one modeling of one system (e.g. world) within another (e.g. a mental logic). I then address the question of learning these representations, drawing on work in theoretical artificial intelligence. The inferential technique allows learners to acquire representations of arbitrary computational complexity yet avoid the primary difficulty that faces learners who operate on spaces of computations, the halting problem. Providing a GPL-licensed implementation of the inference scheme, I then demonstrate how learners could acquire a large variety of mental representations found across human and non-human cognition. In each of these cases, the core symbolic aspect of representation is built out only out of extraordinarily simple and mechanistic dynamics from which the meanings emerge in an interconnected system of concepts. I argue that whatever mental representations are, they must be *like* these kinds of objects, where symbolic meanings for algorithms, structures, and relations arise out of the sub-symbolic dynamics that implement these processes. I then describe how the system can be implemented straightforwardly in existing connectionist frameworks, and discuss broader philosophical implications.

Representation in the Language of Thought

There is a lot going for the theory that human cognition uses, among other things, a structured symbolic system of representation analogous to language. The idea that something like a mental logic describes key cognitive processes dates back at least to Boole (1854), who described his logic as capturing "the laws of thought" and Gottfried Leibniz, who tried to systematize knowledge and reasoning in his own universal formal language, the *Characteristica Universalis*. As a psychological theory, the LOT reached prominence through the works of Jerry Fodor who argues for a compositional system of mental representation that is analogous to human language called a *Language of Thought* (LOT) (Fodor, 1975, 2008). The LOT has had numerous incarnations throughout the history of AI and cognitive science (Newell & Simon, 1976; Penn, Holyoak, & Povinelli, 2008; Fodor, 2008; Kemp, 2012; Goodman, Tenenbaum, & Gerstenberg, 2015; Piantadosi, Tenenbaum, & Goodman, 2016). Most recent versions focus on combining language-like—or program-like—representations with Bayesian probabilistic inference to model concept induction in empirical tasks (Goodman, Tenenbaum, Feldman, & Griffiths, 2008; Goodman, Mansinghka, Roy, Bonawitz, & Tenenbaum, 2008; Yildirim & Jacobs, 2013; Erdogan, Yildirim, & Jacobs, 2015; Goodman et al., 2015; Piantadosi & Jacobs, 2016). If mentalese is like a program, its primitives are humans' most basic mental operations, a view of conceptual representation that has roots and branches in psychology and computer science. Miller and Johnson-Laird (1976) developed a theory of language understanding based on structured, program-like representations. More modern incarnations of conceptual representations can be found in programming languages like Church that aim to capture phenomena like the gradience of concepts through a semantics centered on probability and conditioning (Goodman, Mansinghka, et al., 2008; Goodman et al., 2015). In computer science, the program metaphor has been applied in computational semantics under the name *procedural semantics*, in which representations of linguistic meaning are taken to be programs that compute something about the meaning of the sentence (Woods, 1968; Davies & Isard, 1972; Johnson-Laird, 1977; Woods, 1981). For instance, the meaning of “How many US presidents have had a first name starting with the letter ’T’?” might be captured by a program that searches for an answer to this question in a database. This approach has been elaborated in a variety of modern machine learning models, many of which draw on logical tools closely akin to the LOT (e.g Zettlemoyer & Collins, 2005; Wong & Mooney, 2007; Liang, Jordan, & Klein, 2010; Kwiatkowski, Zettlemoyer, Goldwater, & Steedman, 2010; Kwiatkowski, Goldwater, Zettlemoyer, & Steedman, 2012). Sophisticated AI theories of how knowledge and inference in domains like objects, beliefs, physical reasoning, and time also draw on logical representations mirroring psychological theorizing about the LOT (Davis, 1990).

A LOT would explain some of the richness of human thinking by positing a combinatorial capacity through which a small set of built in cognitive operations can be combined to express new concepts. For instance, in the word learning model of Siskind (1996) the meaning of the word “lift” might be captured as

```
lift(x) = CAUSE(x,GO(x,UP))
```

where `CAUSE`, `GO` and `UP` are simpler conceptual representations—possibly innate—that are composed to express a new word meaning. This compositionality allows theories to posit relatively little innate content, with the heavy lifting of conceptual development (e.g. Piantadosi, Tenenbaum, & Goodman, 2012; Ullman, Goodman, & Tenenbaum, 2012; Mollica & Piantadosi, 2015) accomplished by combining existing operations *productively* in new ways. To discover what composition is “best” to explain their observed data, learners may engage in hypothesis testing or Bayesian inference (Siskind, 1996; Goodman, Tenenbaum, et al., 2008; Piantadosi et al., 2012; Ullman et al., 2012; Mollica & Piantadosi, 2015). The content that a LOT assumes is distinctly symbolic, can be used to generate fundamentally new thoughts, and obeys systematic patterns made explicit in the symbol structures. For example, it would be impossible to think that `x` was lifted without also thinking that `x` was caused to go up. Such compositionality, productivity, and systematicity have been argued to be desirable features of cognitive theories (Fodor & Pylyshyn, 1988), one argued to be lacking in connectionism (for ensuing discussion, see Smolensky, 1988, 1989; Chater & Oaksford, 1990; Fodor & McLaughlin, 1990; Chalmers, 1990; Van Gelder, 1990; Aydede, 1997; Fodor, 1997; Jackendoff, 2002; Van Der Velde & De Kamps, 2006).

However, work on the LOT as a psychological theory has progressed despite a serious problem lurking at its foundation: how is it that symbols themselves come to have meaning? It is far from obvious what would make a symbol `GO` mean go and `CAUSE` mean cause. This is especially troublesome when we recognize that even ordinary concepts like these are notoriously difficult to formalize, perhaps even lacking definitions (Fodor, 1975). Certainly there is nothing inherent in the symbol (the `G` and the `O`) itself that give it this meaning; in some cases the symbols don’t even *refer* to anything external which could ground their meaning either, as is the case for most function words in language (e.g. “for”, “too”, "seven"). This problem seems more pernicious when we consider how what meanings might be to a physical brain. If we look at neural spike trains, for instance, how would we find a meaning like CAUSE?[1]

Meaning through conceptual role

The framework developed here builds off an approach to meaning in philosophy of mind and language known as *conceptual role semantics* (CRS) (Field, 1977; Loar, 1982; Block, 1987; Harman, 1987; Block, 1997; Greenberg & Harman, 2005). CRS which holds that mental tokens get their meaning through their relationship with other symbols, operations, and uses. There is nothing inherently disjunctive about your mental representation of the mental operation OR. What distinguishes it from AND is that the two interact differently with other mental tokens, in particular TRUE and FALSE. The idea extends to more ordinary concepts: a concept of an "accordion" might be inherently about its role in a greater conceptual system, perhaps inseparable from the inferences it licenses about the player, its means of producing sound, its likely origin, etc. An example from Block (1987) is that of learning the system of concepts involved in physics:

> One way to see what the CRS approach comes to is to reflect on how one learned the concepts of elementary physics, or anyway, how I did. When I took my first physics course, I was confronted with quite a bit of new terminology all at once: 'energy', 'momentum', 'acceleration', 'mass', and the like. ... I never learned any definitions of these new terms in terms I already knew. Rather, what I learned was how to use the new terminology—I learned certain relations among the new terms themselves (e.g., the relation between force and mass, neither of which can be defined in old terms), some relations between the new terms and the old terms, and, most importantly, how to generate the right numbers in answers to questions posed in the new terminology.

Indeed, almost everyone would be hard-pressed to *define* a term like "force" in any rigorous way, other than appealing to other terminology like "mass" and "acceleration" (e.g., $f = m \cdot a$). This emphasis on the role of concepts in systems of other concepts leads CRS to be closely related to the Theory Theory in development (see Brigandt, 2004) as well work in psychology emphasizing the role of entire systems of knowledge—*theories*—in conceptualization, categorization, and cognitive development (Carey, 1985; Murphy & Medin, 1985; Wellman & Gelman, 1992; Wisniewski & Medin, 1994; Gopnik & Meltzoff, 1997; Kemp, Tenenbaum, Griffiths, Yamada, & Ueda, 2006; Tenenbaum, Griffiths, & Kemp, 2006; Tenenbaum, Kemp, & Shafto, 2007; Carey, 2009; Kemp, Tenenbaum, Niyogi, & Griffiths, 2010; Ullman et al., 2012; Bonawitz, Schijndel, Friel, & Schulz, 2012; Gopnik & Wellman, 2012). This literature has aimed to computationally model and empirically explore the way in which people learn and use systems of knowledge. These results generally show that the use and acquisition of concepts cannot be studied in isolation—our inferences depend not

[1]The problem is also faced by some connectionist models. For instance, in Rogers and McClelland (2004), a connectionist model of semantics builds in relations (e.g. IS-A, HAS, CAN) and observable attributes (e.g. PRETTY, FLY, SKIN) as activation patterns on individual nodes. There the puzzle remains: what might make it the case that activation in one node *means* PRETTY as opposed to FLY?

only on simple perceptual factors, but the way in which our internal systems of knowledge interrelate.

Block (1987) further argues that CRS satisfies several key desiderata for a theory of mental representation, including its ability to handle truth, reference, meaning, compositionality, and the relativity of meaning. Here I will use the term CRS in a general way, but my implementation will make it unambiguous later. Other authors focus on the *inferential role* of concepts, meaning the way in which they can be used to discover new knowledge. For instance, the concept of conjunction AND is defined by its ability to permit use of the "elimination rule" $P \& Q \rightarrow P$ (Sellars, 1963). In the version of CRS I describe, concepts will be associated with some, but perhaps not all, of these inferential roles, and some other relations that are less obviously inferential. The view that concepts are defined in terms of their relationships to other concepts has close connections to accounts of meaning given in early theories of intelligence (Newell & Simon, 1976), as well as implicit assumptions of computer science. Operations in a computer only come to have meaning in virtue of how they interact with the architecture, memory, and other instructions. For example, nearly all computers represent negative numbers with a *twos' complement* where a number can be negated by swapping the 1s and 0s and adding 1. For instance, in a five-bit processor, we might represent 5 as 00101 and -5 as $11010 + 1 = 11011$. Then, -5 plus one 00001 is $11011 + 00001 = 11100$, which is the representation for -4. This representation is just convention, and equally valid systems have been considered throughout the history of computer science, including using the first bit to represent sign. However, changing the representation of sign would require different algorithms for subtraction and sign changes. This illustrates that the meaning of a representation can only be understood by seeing how it interacts with the other operations.

The primary shortcoming of conceptual role theories as a cognitive account is that they a computational backbone, leaving vagueness about what a "role" might be. When we say that a symbol or a concept gets its meaning from its conceptual role, we must specify what that corresponds to mechanistically and computationally. The lack of computational implementation has given rise to a variety of philosophical debates about what is possible for CRS, but as I argue, at least some of these issues become less problematic once we consider a concrete implementation. The lack of implementations also means that it is difficult to make progress on experimental psychology probing the particular representations and processes of a CRS because there are few ideas about what, formally, a role might be. Addressing this shortcoming is the primary goal of this paper.

Isomorphism and representation

Any CRS theory will have to start by saying *which* mental representations we create and why. Here, it will be assumed that the mental representations we construct are likely to correspond to evolutionarily relevant structures, relations, and dynamics present in the real world.[2] This notion of correspondence between mental representations and the world can be captured with the mathematical idea of *isomorphism*. Roughly, systems X and Y are *isomorphic* if operations in X do "the same thing" as the corresponding operations in Y

[2]Although this notion is controversial—see Hoffman, Singh, and Prakash (2015) and the ensuing commentary.

and vice versa. For instance, the ticking of a second hand is isomorphic to the ticking of an hour hand: both take 60 steps and then loop around to the beginning. How one state leads to the next is "the same" even though the details are different since one ticks every second and the other every minute. Scientific theories form isomorphisms in that they attempt to construct formal systems which capture the key dynamics of the system under study. A simple case to have in mind in science is Newton's laws of gravity, where the behavior of a real physical object is captured by constructing an isomorphism into vectors of real numbers, which themselves represent position, velocity, etc. The dynamics of updating these numbers with Newton's equations is *the same* as updating the real objects.

The notion of isomorphism lies at the heart of many theories of mental representation. Shepard and Chipman (1970) emphasized that while mental representations need not be structurally similar to what they represent, the relationships between internal representations must be "parallel" to the relationships between the real world objects, a pre-cursor to CRS. Gallistel (1998) writes,

> A mental representation is a functioning isomorphism between a set of processes in the brain and a behaviorally important aspect of the world. This way of defining a representation is taken directly from the mathematical definition of a representation. To establish a representation in mathematics is to establish an isomorphism (formal correspondence) between two systems of mathematical investigation (for example, between geometry and algebra) that permits one to use one system to establish truths about the other (as in analytic geometry, where algebraic methods are used to prove geometric theorems).

In this case, mental representations could be used to establish truths about the world without having to alter the world. However, the notion of isomorphism is also deeply connected to the ability of the brain to usefully interact with the world. Conant and Ross Ashby (1970) show that if a system X (e.g. the brain) wishes to control the dynamics of another system Y (e.g. the world), and X does so well (in a precise, information-theoretic sense), then X must have an isomorphism of Y (see Scholten, 2010). This theorem developed out of cybernetics and control theory and is not well-known in cognitive science and neuroscience. Yet, the authors recognized its relevance, noting that the theorem "has the interesting corollary that the living brain, so far as it is to be successful and efficient as a regulator for survival, *must* proceed, in learning, by the formation of a model (or models) of its environment."[3] The centrality of isomorphism in mental representation fits well with CRS since both emphasize that what matters is the relationship between internal states, not the specific objects used in the representation.

But what is mysterious about the brain is that we are able to encode a staggering array of different isomorphisms—from language, to social reasoning, physics, logical deduction, artistic expression, causal understanding, meta-cognition? Such breadth suggests that our conceptual system can support essentially any computation or construct any isomorphism. Moreover, essentially none of this knowledge could possibly be innate because it is so clearly driven by the right set of experiences. Yet, the question of how systems might encode,

[3]I'll conjecture that their proof could be reworked to show that a system which is capable of optimally *predicting* another also must contain an isomorphism since prediction intuitively seems to take as much knowledge as control.

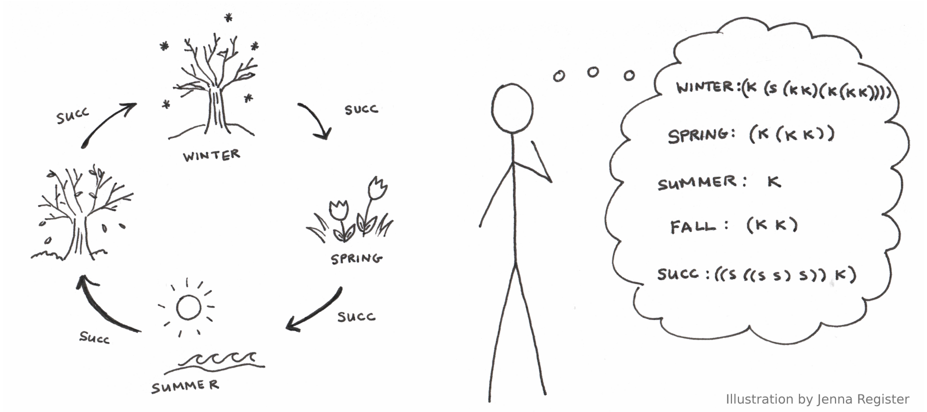


Figure 1. Learners observe relations in the world, like the `succ`essor relationship between seasons. Their goal is to create an internal mental representation which obeys the same dynamics. This is achieved by mapping each observed token to a simple expression written in a universal language whose elements only specify interactions between elements, using a logic for function compositions that is capable of universal computation.

process, and learn isomorphisms—and CRS systems—*in general* has barely been addressed in cognitive science. Indeed, work on the LOT has typically made ad-hoc choices about what primitives should be considered in hypotheses in any given context, thus failing to provide a demonstrably generalized theory of learning that take the breadth of human cognition seriously. The representational system below develops a *universal framework for isomorphism*, a mental system in which we can construct, in principle, a representation of anything else. Unsurprisingly, the existence of such a formalism is closely connected to the existence of universal computation.

The general theory

We are now ready to put together some pieces. The overall setup is illustrated in Figure 1. We assume that learners observe a structure in the world. In this case, the learner sees the circular structure of the seasons, where the successor (`succ`) of spring is summer, the successor of summer is fall, etc. Learners are assumed to have access to this relational information between tokens shown on the left. Their job is to *internalize* (mentally represent) each symbol and relation by mapping symbols to expression in their LOT that obey the right relations, as shown on the right. The mapping will effectively construct an internal isomorphism of the observations, written in the language of mentalese.

A little more concretely, the relations in Figure 1 might be captured with the following facts,

```
(succ winter) → spring
(succ spring) → summer
(succ summer) → fall
(succ fall)   → winter
```

Here, I have written the relations as functions, where for instance the first line means that `succ` is a function applied to `winter`, that returns the value `spring`. The learner's job is to internalize the relations by mapping each of these symbols to a mental LOT expression that obeys *the same* relational structure. So, if `succ`, `winter`, and `spring` get mapped to mental representations ψ_{succ}, ψ_{winter}, and ψ_{spring} respectively, then the first fact means that

$$(\psi_{\texttt{succ}}\ \ \psi_{\texttt{winter}}) \rightarrow \psi_{\texttt{spring}}$$

also holds. Though this statement looks simple, it actually involves some subtlety. It says that whatever internal representation `succ` gets mapped to, this representation must also be able to be used internally as a function. The return value when this mental function is evaluated on ψ_{winter} has to be the same as how `spring` is represented mentally. Each token participates in several roles and must simultaneously yield the correct answer in each, providing a full mental isomorphism of the observed relations.

One might immediately ask why we need anything other than the facts—isn't it enough to know that `(succ spring)` $\rightarrow$ `winter` in that purely symbolic form? For instance, a Prolog program might encode the relations (e.g. ψ_{spring} is a symbol "`SPRING`") and look up facts in a database. It could even be able to answer questions like "The successor of which season is spring?" by compiling these questions into the appropriate database query. Of course, if this worked well, good old fashioned AI would have yielded striking successes. Unfortunately, several limitations of such purely symbolic encoding are clear. First, it is not apparent how looked-up symbols get their meaning, a version of the problem highlighted by Searle (1980)'s Chinese Room. It is not enough to know these symbolic relationships; what matters is the semantic content that they correspond to—cognitive science needs a real theory of meaning. Second, architectures for processing symbols *seem* decidedly unbiological, and the problem of how these symbols may be grounded in a biological system has plagued theories of representation and meaning. Instead, what is needed is a marriage of rules, inference, and abstraction. Third, in many of the cases I'll consider, what matters is not the symbols themselves, but the computations they correspond to. For instance, we might consider a case of a simple algorithm like **repetition**. Your internal concept of repetition must include more than the symbol—it must also encode the process. But what mental representation could encode the general *process* of repetition? Fourth, our cognitive systems must go beyond memorization of facts—we are able to generalize beyond what we have observed, extracting regularities and abstract rules. What might representations be like such that they can allow us to deduce more than what we've already seen? Each of these four goals—meaning, implementation, computation, and induction—can be met with the logical system described below.

The core hypothesis developed in this paper is that the symbols like `succ` and `winter` ↪ get mapped to LOT expressions that correspond only to computational or dynamical objects. Thus, ψ_{succ} is a small dynamical/computational object that, when applied (through function composition) to ψ_{winter} gives us back ψ_{spring} as the return value of the computation. These meanings are specified in a language of pure computational dynamics, absent any additional primitives or meanings. This is shown with the minimalist set of primitives in Figure 1, where each token is mapped to some structure built of **`S`** and **`K`** and whose meaning are purely dynamical (and discussed below). Symbols like `spring` come to have meaning as CRS supposes, by virtue of how they act on other symbols. With the appropriate mapping, learners are able to derive *new* facts by applying their internal expressions to each other in novel ways. As I show, this can give rise to rich systems of knowledge that span classes of computations and permit learners to extend a few simple observations into the domain of richer cognitive theories.

Combinatory logic as a language for universal isomorphism

A mathematical system known as *combinatory logic* provides the formal tool we'll use to construct a universal isomorphism language as a hypothesis about how mentalese may work. Combinatory logic was developed in the early- and mid-1900s in order to allow logicians to work with expressions that did not require variables like "x" and "y" yet had the same expressive power (Hindley & Seldin, 1986). In cognitive research, combinatory logic is primarily seen in formal theories of natural language semantics (Steedman, 2001; Jacobson, 1999). However, its general usefulness is demonstrated by the fact that it was invented at least three independent times by mathematicians, including Moses Schönfinkel, John von Neumann, and Haskell Curry (Cardone & Hindley, 2006). The main advantages of combinatory logic are its simplicity (allowing us to posit very minimal built-in machinery) and its power (allowing us to model symbols, structures, and relations).

This first section will illustrate how combinatory logic can write down a simple function. This illustrates only some of its basic properties, such as its simplicity (involving only two primitives), its ability to handle variables, and its ability to express arbitrary compositions of operations. The more powerful view of combinatory logic comes later, where it is shown how we may use combinatory logic to create a system which is isomorphic to any other computational process using a technique known as Church encoding.

A very brief introduction to combinatory logic

To illustrate the basics of combinatory logic, consider the simple function definition,

$$f(x) = x + 1. \tag{1}$$

The challenge with expressions like (1) is that the use of a variable x adds bookkeeping to a computational system because one has to keep track of what variables are allowed where. Compare (1) to a function of two variables $g(x, y) = \ldots$. When we define `f`, we are permitted to use x. When we define g, we are permitted to use both x and y. But when we define `f`, it would be nonsensical to use y, assuming y is not defined elsewhere. Analogously in a programming language—or cognitive/logical theories that look like programs—we can only use variables that are defined in the appropriate context (scope). The syntax of what symbols are allowed changes in different places in a representation.[4] What logician Moses Schönfinkel discovered was that this situation could be avoided by using *combinators* to glue together the primitive components +, and 1 without ever explicitly creating a variable x. A combinator is a higher-order function (a function whose arguments are functions) that, in this case, routes arguments to the correct places. For instance using := to denote a definition, let

```
f := (S + (K 1))
```

[4]In combinatory logic's rival, lambda calculus (Church, 1936), much formal machinery is spent ensuring that variable names are distinct and only used in the appropriate places, and that substitution does not incorrectly handle variable names.

define `f` in terms of other functions **S** and **K**, in addition to the operator + and the number 1. Notably there is no x in the above expression for `f`, even though `f` does take an argument. The functions **S**&**K** have very simple definitions:

```
(K x y)   → x
(S x y z)  → ((x z) (y z))
```

Here, the arrow (→) indicates a process of evaluation, or moving one step forward in the computation. The combinator **K** takes two arguments `x` and `y` and ignores `y`, a constant (`k`) function. **S** is a function of three arguments, `x`, `y`, and `z`, that essentially passes `z` to each of `x` and `y` before composing the two results.[5] In this notation, if a function does not have enough arguments it may take the next one in line. For instance in `((K x) y)` the **K** only has one argument. But it can grab the `y` as its second argument, meaning that computation proceeds,

```
((K x) y) → (K x y) → x
```

Doing so must respect the grouping of terms, so that `((K x)(y z))` becomes `(K x (y z` ↪ `))`. This capacity to take the next argument is known in logic as *currying*, although Curry attributed it to Schönfinkel, and it was more likely first invented by Frege (Cardone & Hindley, 2006). Together, **S**&**K** and currying define a logical system that is much more powerful than it first appears.

To see how the combinator definition of `f` works, we can apply `f` to an argument. For instance, if we evaluate `f` on the number 7, we get can substitute in the definition of `f` into the expression `(f 7)`:

```
(f 7) :=  ((S + (K 1)) 7)     ; Definition of f
→ (S + (K 1) 7)              ; Currying
→ ((+ 7) ((K 1) 7))          ; Definition of S
→ (+ 7 ((K 1) 7))            ; Currying
→ (+ 7 (K 1 7))              ; Currying
→ (+ 7 1)                    ; Definition of K
```

Essentially what has happened is that **S**&**K** have shuttled 7 around to the places where x would have appeared. They have done so merely by their compositional structure and definitions, without ever requiring the variable x to be written. Schönfinkel—and other independent discoverers of combinatory logic—proved the non-obvious fact that *any* function composition could be expressed this way. In other words, compositional functions like **S**&**K** can allow computations to take place without needing variables.

Terminologically, the process of applying the rules of combinatory logic is known as *reduction*. The question of whether a computation halts is equivalent to whether or not reduction leads to a *normal form* in which none of the combinators have enough arguments to continue reduction.

In terms of computational power, combinatory logic is equivalent to lambda calculus (see Hindley & Seldin, 1986), both of which are capable of expressing any computation through function composition (Turing, 1937). This means that any typical program (e.g. in Python or C++) can be reduced to a composition of these combinators. Equivalently, any system that implements these very simple rules for **S**&**K** is, potentially, as powerful as any computer. This is a remarkable result in mathematical logic because it means that

[5] In other notation, **S** could be defined as **S**$(x, y, z) := x(z, y(z))$.

computation can be expressed with the simplest syntax imaginable, compositions of **S**&**K** with no extra variables or syntactic operations. Evaluation is equally simple and requires no special machinery beyond the ability to perform two tree transformations. This uniformity and simplicity of syntax opens the door for straightforward implementation in physical or biological systems.

Church Encoding

The example above uses primitive operations like + and objects like the number 1. It therefore fits well within the traditional LOT view where mental representations correspond to compositions of intrinsically meaningful elements. The primary point of this paper, however, is to argue that the right metaphor for mental representations is *not* structures like (1) or its combinator version, but rather structures *without any cognitive primitives at all*—that is, structures that contain only combinators and no additional operations (like `1` or `+`).

The technique behind this is known as *Church encoding*, and it corresponds to building a structure in a purely logical system that obeys the correct relations. The idea is that if symbols and operations are encoded as pure combinator structures, they may act on each other to produce equivalent algorithms to those that act on numbers, boolean operators, trees, or any other formal structure. As Pierce (2002) writes,

> ... suppose we have a program that does some complicated calculation with numbers to yield a boolean result. If we replace all the numbers and arithmetic operations with [combinator]-terms representing them and evaluate the program, we will get the same result. Thus, in terms of their effects on the overall result of programs, there is no observable difference between the real numbers and their Church-[encoded]numeral representations.

A simple, yet philosophically profound, demonstration is to construct a combinator structure that implements **Boolean logic**. One possible way to do this is to define

```
true  := (K K)
false := K
and   := ((S (S (S S))) (K (K K)))
or    := ((S S) (K (K K)))
not   := ((S ((S K) S)) (K K))
```

Defined this way, these combinator structures obey the laws of Boolean logic: `(not true)` → `false`, and `(or true false)` → `true`, etc. The meaning of mental symbols like `true` and `not` is given *entirely* in terms of how these little algorithmic chunks operate on each other. To illustrate, the latter computation would proceed as

```
(or true false) = (((S S) (K (K K)))  (K K)  K)          ; Definition of or,
   ↪ true, false
→ (((S S) (K (K K)) (K K))  K)       ; Currying rule
→ ((S S (K (K K)) (K K)) K)          ; Currying rule twice
→ ((S (K K) ((K (K K)) (K K))) K)    ; Definition of S
→ ((S (K K) (K (K K) (K K))) K)      ; Currying
→ ((S (K K) (K K)) K)                ; Definition of K
→ (S (K K) (K K) K)                  ; Currying
→ ((K K) K ((K K) K))                ; Definition of S
→ ((K K K) ((K K) K))              ; Currying
→ (K ((K K) K))                    ; Definition of K
→ (K (K K K))                      ; Currying
→ (K K)                            ; Definition of K
```

resulting in an expression which is the same as the one for true! Skeptical or neurotic readers may also verify other relations, like that `(and true true)` → `true` and `(or (not` ↪ `false)false)` → `true`, etc.

The Church encoding has essentially tricked **S&K**'s boring default dynamics into doing something useful, implementing a theory of simple boolean logic. This is a CRS because the symbols have no intrinsic meaning beyond that which is specified by their dynamics and interaction. As shown above, the ability to use **S&K** to perform useful computations is very general, allowing us to encode complex data types, operations, and a huge variety of other logical systems. Appendix A sketches a simple proof of conditions under which combinatory logic is capable of representing *any* consistent set of facts or relations, making it truly a universal isomorphism language. It is this combination of generality and simplicity that makes combinatory logic a desirable language for formalizing notions of computation, conceptual role, and cognition.

An inferential theory from the probabilistic LOT

The capacity to represent anything is, of course, not enough. Our cognitive theory must also have the ability to construct the right particular representations when data is observed. The data that we will consider is sets of *base facts* like those shown in Figure 1, `(succ winter)` → `spring`, etc. These facts may be viewed as structured representations of perceptual observations—for instance, the observation that some season (`spring`) comes after another (`winter`). Note, though, that the meanings of these symbols are not specified by these facts; all we know is that `spring` (whatever that is) comes after (whatever that is) the season `winter` (whatever that is). Apart from any perceptual links, that knowledge is structurally no different from `(father jim)` → `billy`. Because these symbols do not yet have meanings, knowledge of the base facts is much like knowledge of a *placeholder structure* (Carey, 2009), or concepts whose meanings yet to be filled in, even though some of their conceptual role is known.

The goal of the learner is to assign each symbol a combinator structure so that the structures in the base facts are satisfied. For this one rule `(succ winter)` → `spring` we could assign `succ := (K S)`, `winter := I` and `spring := S` since then

```
(succ winter) := ((K S) I) → (K S I) → S = spring
```

Only some mappings of symbols to strings will be valid. For instance, if `spring := I` instead, we'd have that

```
(succ winter) := ((K S) I) → (K S I) → S ≠ spring.
```

The trick is to find a mapping of symbols to combinators that satisfies *all of the facts simultaneously*. Such a solution provides an internal model—a Church encoding—whose computational semantics captures the observed facts. Often, the mapping of symbols to combinators will often be required to be *unique*, meaning that we can always tell which symbol a combinator structure stands for. Once symbols are mapped to combinators satisfying the observed base facts, learners or reasoners may derive new generalizations that go beyond these facts.

The choice of which combinator each symbol should be mapped to is made using inductive ideas from the LOT. Human learners prefer to induce hypotheses that have a shorter description length in logic (Feldman, 2000, 2003a; Goodman, Tenenbaum, et al., 2008), with simplicity preferences perhaps a governing principle of cognitive systems (Feldman, 2003b; Chater & Vitányi, 2003). Simplicity-based preferences have been used to structure the priors in standard LOT models (Goodman, Tenenbaum, et al., 2008; Katz, Goodman, Kersting, Kemp, & Tenenbaum, 2008; Ullman et al., 2012; Piantadosi et al., 2012; Piantadosi, Goodman, & Tenenbaum, under review; Kemp, 2012; Yildirim & Jacobs, 2014; Erdogan et al., 2015), and has close connections to the idea of *minimum description lengths* (Grünwald, 2007). The problem with theories based on description length is that they can easily run into computability problems: short programs or logical expressions often do not halt computation[6] meaning that we may not be able to even evaluate every given logical hypothesis.

A solution is to instead base our preferences on *running time* (or *evaluation time*), for instance assigning a prior to a hypothesis h that is proportional to $2^{-t(h)}$ where $t(h)$ is the amount of time it takes h to halt evaluation. In combinatory logic, this means that the amount of time it runs until combinators are in a normal form. This idea has been developed in theories of artificial intelligence (Hutter, 2005; Schmidhuber, 2007); related ideas can be found as a measure of complexity (Bennett, 1995). Running time allows learners to operate in Turing-complete spaces because expressions that do not halt will have zero ($2^{-\infty}$) probability. As they run, their probability drops, meaning that they can effectively be pruned out of searches before seeing whether they eventually halt, a fact that can also be used in Markov-Chain Monte-Carlo techniques like the Metropolis Hastings algorithm often used in LOT models (Goodman, Tenenbaum, et al., 2008).

Since so much other work has explored the probabilistic details of LOT theories, and I intend to provide a simple demonstration, I'll make two simplifying assumptions in this paper. First, I assume that learners want only the fastest-running combinator string which describes their data (e.g. ignoring the gradience of fully Bayesian accounts). Second, it will be assumed that only theories that are consistent with the data are considered. This will therefore assume that leaner's data is noise-free, although the general inferential mechanisms can readily be extended to noisy data (see LOT citations above).

Details of the implementation

The problem of finding a concise mapping of the symbols to combinators that obey these laws is solved here using a custom Scheme implementation named *ChurIso* (pronounced like the sausage "chorizo") and available under a GPL license[7]. The implementation was provided with base facts and searched for mappings from symbols to combinators that satisfies those constraints under the combinator dynamics defined above. The general idea of finding a formal internal representation satisfying observed relations has close connections to model theory (Ebbinghaus & Flum, 2005; Libkin, 2013), as well as the so-

[6]One simple "non-halting" combinator is `(S (S K K) (S K K) (S (S K K) (S K K)))`.
[7]https://github.com/piantado/ChurIso

lution of constraint satisfaction problems specified by logical formulas (*satisfiability modulo theories*) (Davis & Putnam, 1960; Nieuwenhuis, Oliveras, & Tinelli, 2006). Among all mappings of symbols to combinators that are consistent with the base facts, those with the fastest running time are preferred. The search is limited by length as well for computational tractability, but among those found, the fastest-running combinator structure is preferred.

The implementation uses a backtracking algorithm that exhaustively tries pairing symbols and combinator structures (ordered in terms of increasing description-length complexity), rejecting a partial solution if it is found to violate a constraint. Several optimizations are provided in the implementation. First, the set of combinators considered for each symbol can be limited to those already in normal form to avoid re-searching equivalent combinators. Second, the algorithm uses a simple form of constraint propagation in order to rapidly reject assignments of symbols to combinator strings that would violate a later constraint. For instance, if a constraint says that `(f x)` must reduce to `y`, and `f` and `x` are determined, then the resulting value is pushed as the assignment for `y`. In order to explore the space, ChurIso also allows us to define and include other combinators either as base primitives or as structures derived from **`S&K`**. The results in this paper use the search algorithm including several other standard combinators (`B`, `C`, **`I`**) to increase the search effectiveness, but each is converted to **`S&K`** in evaluating a potential hypothesis.

In a testament to the simplicity and parsimony of combinatory logic, the full implementation requires only a few hundred lines of code, including algorithms for reading the base facts, searching for constraints, and evaluating combinator structures. Ediger (2011) provides an independent combinatory logic implementation that includes several abstraction algorithms and was used to test ChurIso.

Applications to cognitive domains

This section presents a number of examples of using the inferential setup to discover combinator structures for a variety of domains. In each example, I will provide the base facts and then the fastest running combinator structure (Church encoding of the base facts) that was discovered by ChurIso. The examples have been chosen to illustrate a variety of different domains that have been emphasized in human and animal psychology. The first section shows that theories can represent or encode relational structures. The second second examines conceptual structures that involve *generalization*, meaning that we are primarily interested in how the combinator structure extends to compute new relations not in set of base facts. In each of these cases, the generalization fits simple intuitions about and permits derivation of new knowledge in the form of "correct" predictions about unobserved new structures. The third section look at combinatory logic to represent new *computations* in the form of components that could be used in new mental algorithms, and the fourth uses these computational processes to approximate some formal languages.

Representation

Table 1 shows five domains with very different structural properties and how they may be represented with **`S&K`**. The middle column shows the base facts that were provided to ChurIso and the right hand column shows the most efficient combinator structure. The **seasons** example show a circular system of labeling, where the successor (`succ`) of each season loops around in a Mod-4 type of system. The **1, 2, Many** concept where there is similarly a successor, but the successor of any number above `two` is just the token `many`, a counting system found in many languages of the world. **Roshambo** (also called "Rock, paper, scissors") is an interesting case where each object represents a function that operates on each other object, and returns an outcome in a discrete set ("rock" beats "scissors", etc.). This game has been studied in primate neuroscience (Abe & Lee, 2011). This illustrates a type of circular dominance structure, but one which is dependent on which object is considered the operator (e.g. first in the parentheses) and which is the operand. The **family tree** example shows a case where simple relations like `mother` and `husband` can be defined and are functions that yield the correct individual. Realization of this representation therefore corresponds to encoding a particular family tree (for work on learning the general case with LOT theories, see Katz et al. (2008); Mollica and Piantadosi (2015)). Note that in all examples, the combinator structures ChurIso discovers shouldn't be intuitively obvious to us—these combinators structures are not the symbols we are used to thinking about (like `father` and `many`), certainly not in conscious awareness. Instead, the base facts should sound obvious; the **S**&**K** structures are the stuff out of which the symbols in the base facts are made.

Generalization

The examples in Table 1 mainly shows how learners might memorize facts and relations using **S**&**K**. But equally or more important to cognitive systems is *generalization*: given some data, can we learn a representation whose properties allow us to infer new and unseen facts? What this means in the context of ChurIso is that we are able to form new combinations of functions—those whose outcome is not specified in the base facts. Table 2-3 shows some examples. The first of these is a representation of a **singular/plural** system like those found in natural language. Here, there is a relation `marker` that takes some number of arguments `item`, `item`, etc. and returns `singular` if it receives one argument, `plural` if it receives more than one. This requires generalization because the base facts only show the output for up to four items. However, the learned representation is enough to go further to any number of items, outside of the base facts. For instance,

```
(marker item item item item item item item) → ((S (S K)) (S (S K))) = plural
```

Intuitively, the first time `marker` is applied to `item`, we get

```
(marker item) → ((S (K (S K K))) (S (S K))) = singular
```

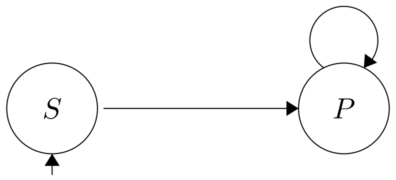
When this is applied to another `item`, you get the expression for `plural`:

```
(marker item item) = ((marker item) item) → (singular item) → plural
```

And then `plural` has the property that it returns itself when given one more item:

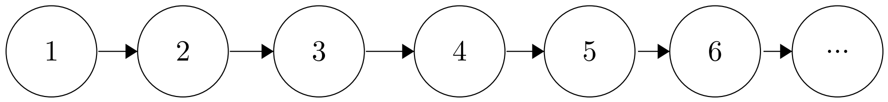
```
(plural item) → plural
```

So, `plural` is a "fixed point" for further applications of `item`, allowing it to generalize to any number of arguments. In other words, what ChurIso discovers is a system that is functionally equivalent to a simple finite-state machine:



Note that this FSM is not an explicitly-specified hypothesis that learners have, but only emerges implicitly through the appropriate composition of **`S&K`**.

The next domain, **number**, shows a generalization that builds an infinite structure. Intuitively, the learner is given a `succ`essor relationship between the first few words. The critical question is whether this is enough to learn a **`S&K`** structure for `succ` that will continue to generalize *beyond* the meaning of `four`. The results show that it is: the form of `succ` that is learned is essentially one that builds a "larger" representation for each successive number. The way this works is extremely simple: **`K`** requires more than one argument. But, the structure represented in the relation `(succ x)` for `x=one, two, ...` only provides it a single argument (`x`). Thus, the n'th number in this Church encoding is the one that needs n more arguments to successfully evaluate (or reduce to nothing). This will make it such that `(succ four)` is a new concept (representation) and `(succ (succ four))` is yet another, generalizing infinitely to infinitely many numbers. In this case, **`S&K`** create from the three base facts a structure isomorphic to the natural numbers,



The assumed base facts correspond to the kind of evidence that might be available to learners of counting (Carey, 2009). This provides a close theory to Piantadosi et al. (2012)'s LOT model of number acquisition, which was created to solve the inductive riddle posed by Rips and colleagues (Rips, Asmuth, & Bloomfield, 2006, 2008; Rips, Bloomfield, & Asmuth, 2008) about what might constrain children's generalization in learning number. The difference is that the Piantadosi et al. (2012)'s LOT representations were based in primitive cognitive content, like an ability to represent sets and perform operations on them. Here, the learning is not a counting algorithm, but rather an internal conceptual structure that is generative of the concepts themselves, providing a possible answer to the question of where the structure itself may come from (see Rips, Asmuth, & Bloomfield, 2013). It is interesting to contrast **number**, **1, 2, Many**, and **seasons**. In each of these, there is a "successor" function, but which function is learned depends on the structure of the base facts. This means that notions like "successorship" cannot be defined narrowly by the relationship between an a few elements, but will critically rely on the role this concept plays in a larger collection of operations.

The **dominance** concept in Table 3 shows another interesting case of generalization. Dominance structures are common in animal cognition (see, e.g., Drews, 1993). Grosenick, Clement, and Fernald (2007) show that fish can make transitive inferences consistent with a dominance hierarchy: if they observe $a \succ b$ and $b \succ c$, then they know that $a \succ c$, where $\succ$ is a relation specifying who dominates in a pairwise interaction. The base facts for the **dominance** encoding slightly more information, corresponding to almost all of the dominance relationships between some mental tokens `a`, `b`, `c`, and `d`.[8] This example illustrates another feature of ChurIso: we are able to specify constraints in terms of evaluations *not* yielding some relations. So for instance,

```
(dom b a) ↛ True
```

means that `(dom b a)` evaluates to something other than `True`. This relaxation of the constraints to only partially-specified often helps to learn more concise representations in **S&K**. Critically the relation between `a` and `d` is not specified in the base facts. Note that this relation *could* be anything and any possible value could be encoded by **S&K**. The simplicity bias of the inference, however, prefers combinator structures for these symbols such that the *unseen* relation `(dom a d)` → `True` but `(dom d a)` does not. Thus, the **S&K** encoding of the base facts gives learners an internal representation that automatically generalizes to an unseen relation.

The **magnetism** example is motivated by Ullman et al. (2012)'s model studying the learning of entire systems of knowledge (*theories*) in conceptual development. In magnetism, we know about different kinds of materials (positive, negative, non-magnetic) and that these follow some simple relationships, like that positives attract negatives and that positives repel each other, etc. The **magnetism** example provides base facts giving the pairwise interaction of two positive two positives (`p1`, `p2`) and two negatives (`n1`, `n2`). But from the point of view of the inference, these four symbols are not categorized into "positives" and "negatives", they are just arbitrary symbols. In this example, I have also dropped the uniqueness requirement to allow grouping of these symbols into "types", as shown by their learned combinator structures with the `pi` getting mapped to the same structure and the `ni` getting mapped to a different one. To test generalization, we can provide the model with one single additional fact, that `n1` and `x` attract each other. The model automatically infers that `x` has the same **S&K** structure as `p1` and `p2`, meaning that it learns from a single observation of it is a "positive", including all of the associated predictions such as that `(attract n1 x)` → `True`.

Computational process

The examples above respectively show computation and generalization, but they do not illustrate one of the most remarkable properties of thinking—we appear able to discover a wide variety of computational *processes*. The concepts in Table 4 are ones that implement some simple and intuitive algorithmic components. Here, I have introduced some new symbols to the base facts, f, x, and y. These are treated as universally quantified variables, meaning that the constraint must hold for *all* values (combinator expressions) they can take. The learning model's discovery of how to encode these facts corresponds to the creation of fundamental algorithmic representations using only the facts' simple description of what the algorithm must do.

An example is provided by **if-else**. A convenient feature of many computational systems it that when they reach a conditional branch ("if" statement), they only have to evaluate the corresponding branch of the program. The shown base facts make `if-else`

[8]Intuitively, more data is needed than in the simple Fish transitive inference cases because the **S&K** model does not inherently know it is dealing with a dominance hierarchy. Cases of dominance hierarchies in animal cognition may have a better chance of being innate, or at least higher prior than other alternatives.

return `x` if it receives a true first argument and `y` otherwise, regardless of what `x` and `y` happen to be. Even though conditional branching is a basic computation, it can be learned from even more primitive components **S&K**.

The **identity** example illustrates the distinction between implicit and explicit knowledge in the system. We can define `identity := (S K K)` so,

```
(identity x) = ((S K K) x) → (S K K x) → ((K x) (K x)) → (K x (K x)) → x.
```

It may be surprising that we could construct a cognitive system without any notion of identity. Surely to even perform a computation on a representation `x`, the identity of `x` must be respected! In **S&K**, this is true in one sense: the default dynamics of **S** and **K** do respect representational identity. But in another sense, such a system comes with no "built in" representation of a function which is the identity function.

A more complex example can be found in the example of **repetition**. Here, we seek a function `repeat` that takes two arguments f and x and calls f twice on x. Humans clearly have cognitive representations of a concept like *repeat*ing a computation; "again" is an early-learned word, and the general concept of repetition is often marked morphologically in the world's languages with reduplication. As is suggested by the preceding examples, the concept of repetition need not be assumed by a computational system.

Related to repetition, or doing an operation "again" is the ability to represent **recursion**, a computational ability that has been hypothesized to be the key defining characteristic of human cognition (Hauser, Chomsky, & Fitch, 2002). One example of how to implement recursion in combinatory logic is the *Y-combinator*,

```
Y = (S (K (S I I)) (S (S (K S) K) (K (S I I)))),
```

a function famous enough in mathematical logic to have been the target of at least one logician's arm tattoo. Like other concepts, the Y-combinator can be built only from **S&K**. It works by "making" a function recursive, passing the function to itself as an argument. The details of this clever mechanism are beyond the scope of this paper (see Pierce, 2002). One challenge in learning `Y` is that by definition it has no normal form when applied to a function. To address this, we introduce a new kind of constraint $\dashrightarrow$, which holds true if the *partial* evaluation trace of the left and right hand sides yield expressions that are equal to a given fixed constant depth. To learn recursion, we that applying `Y` to `f` is the same as applying `f` to this expression itself,

```
(Y f)  ⇢   (f (Y f))
```

Neither side reduces to a normal form, but the special equality symbol means that when we run both sides, we get out the same structure, which in this case happens to be the (infinite) recursion of f composed with itself,

```
(f (f (f (f ...))))
```

ChurIso learns a slightly longer form than the typical Y-combinator due to the details of its search procedure (for the most concise recursive combinator possible, see Tromp, 2007). The ability to represent `Y` permits us to capture algorithms, some of which may never halt. For instance, if we apply `Y` top the definition of successor from the **number** example, we get back the concept of that counts forever, continuously adding one to its its result: `(Y succ)`.

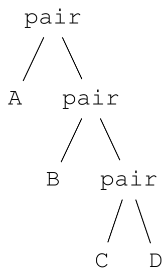
The **mutual recursion** case shows a recursive operator of two functions, known as the `Y*`-combinator, that yields an infinite alternating composition of f and g,

```
(f (g (f (g (f (g ...))))))
```

This is the analog of the Y-combinator but for mutually recursive functions–where f is defined in terms of g and g is defined in terms of f. This illustrates that even more sophisticated kinds of algorithmic processes can be discovered and implemented in **S&K**.

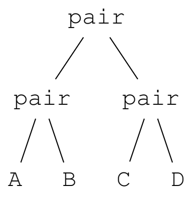
From surprisingly small base facts, ChurIso is also able to discover primitives `first` ↪ , `rest`, and `pair`, corresponding to the structure-building operations with memory. The arguments for `pair` are "remembered" by the combinator structure until they are later accessed by either `first` or `rest`. As a result, they can build common data structures. For instance, a list may be constructed by combining `pair`:

```
L = (pair A (pair B (pair C D)))
```



Or, a binary tree may be encoded,

```
L = (pair (pair A B) (pair C D))
```



An element such as `C` may then be accessed `(first (rest T))`, the first element of the second grouping in the tree. These data structures are so foundational that they form the foundational built-in data type in programming languages like Scheme and Lisp (where they are called *car*, *cdr*, and *cons* for historical reasons), and thus support a huge variety of other data structures and algorithms (Abelson & Sussman, 1996; Okasaki, 1999). By showing how speakers might internalize these concepts, we can therefore demonstrate in principle how many algorithms and data structures could be represented as well.

Formal languages

One especially interesting case to consider is how **S&K** handles concepts that correspond to (potentially) infinite sets of strings, or *formal languages*. Theoretical distinctions between classes of formal languages form the basis of computational theories of human language (e.g. Chomsky, 1956, 1957) as well as computation itself (see Hopcroft, Motwani, & Ullman, 1979). To implement each, Table 5 provides base facts giving the transitions between computational states for processing languages. The **regular** language provides the transition table for a simple finite-state machine that recognizes the language $\{ab, abab, ababa, \dots\}$. The **existential** one also describes a finite state machine that can implement first-order quantification, an association potentially useful in natural language semantics (van Benthem, 1984; Mostowski, 1998; Tiede, 1999; Florêncio, 2002; Gierasimczuk, 2007).

The most interesting example is provided by **context-free**, which is a language $\{ab, aabb, aaabbb, \ldots\}$ that provably cannot be expressed with a regular language (finite-state machine). Instead, the learned mapping essentially implements a computational device with an infinite number of states from the base facts. For instance, the state after observing 1, 2, and 3 `a`s are,

```
got_a    := ((S (K K)) S)
got_aa   := ((S (K K)) (S (K K) S))
got_aaa  := ((S (K K)) (S (K K) (S (K K) S)))
got_aaaa := ((S (K K)) (S (K K) (S (K K) (S (K K) S))))
```

Each additional `a` adds to this structure. Then, each incoming `b` removes from it

```
(b got_aaaa) = want_bbb = (K (K (K (K (S S)))))
(b got_aaa)  = want_bb  = (K (K (K (S S))))
(b got_aa)   = want_b   = (K (K (S S)))
(b want_b)   = accept   = (K (S S))
```

This works precisely like a *stack* in a parser, even though such a stack is not explicitly encoded into **S&K** or the base facts. Thus, this mapping generalizes infinitely to strings of arbitrary length, far beyond the input base facts' length of four (Note that the base facts and combinators ensure correct recognition, but do not guarantee correct rejection).

Finally, the **finite** example shows an encoding of the set of strings of the letters "m", "a", "n" and space ("_") that form valid English words, $\{a, man, am, an, mam\}$. These can be encoded by assigning each character a combinator structure, but the resulting structures are quite complex. Note, too, that these base facts do not guarantee correct generalization to longer character sequences. This example illustrates that while Church encoding can represent such information, it is unwieldy for rote memorization. Church encoding is more likely to be useful for algorithmic processes and conceptual systems with many patterns. Memorized facts (or sets) may instead rely on specialized systems of memory representation.

The ability to represent formal languages like these is important because they correspond to provably different levels of computational power, showing that a single system for learning and representation across these levels is a defining strength of this approach (for for LOT work along these lines, see Yang & Piantadosi (in prep); for language learning on Turing-complete spaces in general, see Chater and Vitányi (2007); Hsu and Chater (2010); Hsu, Chater, and Vitányi (2011)). In the examples, we have taught ChurIso the full algorithm by showing it a few steps from which it generalizes the appropriate algorithm. This ability demonstrates the induction of a novel dynamical system from a few simple observations, work in many ways reminiscent of encoding parsing in continuous dynamical systems (Tabor, Juliano, & Tanenhaus, 1997).

Summary of computational results

The results of this section have shown that learners can *in principle* start with only representations of **S&K** and construct much richer types of knowledge. Not only can they represent structured knowledge, by doing so they permit derivation of fundamentally new knowledge and types of information processing. The ability of a simple search algorithm to actually discover these kinds of representations shows that the resulting representational and inductive system can "really work" on a wide variety of domains. However, the main contribution of this work is the general lessons that we can extract from considering systems like `S&K`.

Mental representations are *like* Combinatory Logic (LCL)

My intention is not to claim that combinatory logic is *the* solution to mental representation—it would be quite lucky if logicians of the early 19th century happened to hit on the right theory of how a biological system works! Rather, I see it as a metaphor with some of the right properties—whatever mental representations are, they must be similar to combinatory logic in a number of key ways. I will refer to the more general form of the theory as ***like* Combinatory-Logic**, or LCL, and describe some of its core components.

LCL theories have no cognitive primitives. The primitives used in LCL theories (like `S&K`) specify only the dynamical properties of a representation—how each structure interacts with any other. LCL therefore has no built-in *cognitive* representations, or symbols like `CAUSE` and `True`. This is the primary difference between LCL and LOT theories, whose bread and butter is meaningful components with intrinsic meaning. The lack of these operations is beneficial because LCL therefore leaves no lingering questions about how mental tokens may come to have meaning. The challenge for LCL is then to eventually specify how a token such as `CAUSE` comes to have its meaning by formalizing the necessary and sufficient relations to other concepts that fully characterize its semantics.

LCL theories are universal and Turing-complete. Though it is not widely appreciated in cognitive science or philosophy of mind, humans excel at learning, internalizing, creating, and communicating algorithmic processes of incredible complexity. This is most apparent in domains of expertise—a good car mechanic or numerical analyst has a remarkable level of technical and computational knowledge, including not only domain-specific facts, but knowledge of specific algorithms, processes, causal pathways, and causal interventions. The mystery for development is to discover how to start with what a baby might know and build the genuinely complex algorithms and representations that adults know. The power of LCL systems come from starting with a small basis of computational elements that have the capacity to express arbitrary computations, and applying a powerful learning theory that can operate on such spaces.

LCL theories are compositional. The compositionality of natural language and natural thinking indicates that mental representations must themselves support composition (Fodor, 1975). Semantic formalisms in language (e.g Heim & Kratzer, 1998; Steedman, 2001; Blackburn & Bos, 2005) rely centrally on compositionality, dating back to Frege (1892). It is no accident that these theories formalize meaning through function composition, using a system (λ-*calculus*) that is formally equivalent to combinatory logic. The inherently compositional architecture of LCL contrasts with Turing machines and von Neumann architectures, which have been dominant conceptual frameworks primarily because they are easy for us to conceptualize and implement in hardware. But when we consider the rampant compositionality of thought, *a computational formalism based in composition* becomes a more plausible starting point.

LCL theories are structured. As with compositionality, the structure apparent in human language and thought seems to motivate a representational theory that incorporates structure, like the structures of LCL (`(S (K S))` is a different concept/computation than `((S K) S)`, even though they are composed of the same elements in the same order). Structure-sensitivity is also a central feature of thought since thoughts can be composed of the same elements but differ in meaning due to their compositional structure (e.g. "Bill loves Mary" compared to "Mary loves Bill").

Indeed, LCL's emphasis on isomorphism aligns it closely with the literature on *structure mapping* (Gentner, 1983; Falkenhainer, Forbus, & Gentner, 1986; Gentner & Markman, 1997; French, 2002), where the key operation is the construction of a correspondence between two otherwise separate systems. For instance, we might consider an alignment between the solar system and the Bohr model of the atom, where the sun corresponds to the nucleus and the planets to electrons. The correspondence is relation preserving in that a relation like `orbits(planets,sun)` holds true when its arguments are mapped into the domain of atoms, `orbits(electrons,nucleus)`. What structure mapping literature does not emphasize, however, is that the systems being aligned are sometimes dynamic and computational, rather than purely structural (isomorphism is a mapping of dynamics). Moreover, work on structure mapping has focused on high-level tasks like computation of analogy, rather than the sense in which relational isomorphisms might be the foundation of meaning itself.

LCL also shares much motivation and machinery with the literature on *structure learning*, which has aimed to explain how learners might discover latent structured representations, which then can guide further inference and learning. Kemp and Tenenbaum (2008) show how learners could use statistical inference to discover the appropriate mental representation in a *universal* graph grammar capable of generating any structure. They show how learners could discover representations appropriate to many sub-domains such as phylogenetic trees for animal features, or the left-right spectrum seen in supreme court judgments. A limitation of that work is that it focuses on learning graph structures, not computational objects that can capture internal processes and algorithms, like LCL.

LCL theories handle abstraction. Combinatory logic was created as a system to allow *abstraction* with a simple, uniform syntax that avoids difficulties with handling variables. Marcus (2003) considers training data like "A rose is a rose", "A frog is a frog", "A blicket is a _____ ?" The intuition is that "blicket" is a natural response, even though we do not know what a blicket is. This means that we must have some system capable of remembering the symbol in the first slot of "A _____ is a _____" and filling it in the second slot. This problem more generally faces systems tasked with understanding and processing language (Jackendoff, 2002). Sub-symbolic approaches have explored a variety of architectural solutions to this variable binding problem (Hadley, 2009), including those based on temporal synchrony (Shastri, Ajjanagadde, Bonatti, & Lange, 1996), tensor products (Smolensky, 1990; Smolensky & Legendre, 2006), neural blackboard architectures (Van Der Velde & De Kamps, 2006), and vector symbolic architectures (Gayler, 2004, 2006). Marcus (2003) argues for explicitly variables in the sense of symbolic programming languages like Lisp; some work in the probabilistic LOT has explored how symbolic architectures might handle variables (Overlan, Jacobs, & Piantadosi, 2016) and how abstraction helps inductive inference (Goodman, Ullman, & Tenenbaum, 2011).

Unfortunately, the debate about explicit variables has been completely misled by the notation that happens to be used in computer science and algebra. Table 6 shows various levels of abstraction for a simple function f, none of which involve variables when expressed with **S&K**. The top row is a function of no arguments that always computes $1 + 4$. The next rows shows a function of one variable, x; the third adds its two arguments; the fourth row shows a highly abstract function that applies an operation (perhaps +) to its two arguments x and y. In none of these abstract functions do the arguments appear explicitly, meaning that abstraction can be captured without variables.

LCL theories permit construction of systems of knowledge. It is absolutely central to LCL systems that the meaning of a representation can only be defined by the role it plays in an interconnected system of knowledge. There is no sense in which any of the combinator structures mean anything in isolation. Even for a single domain like **number**, the most efficient mapping to combinators will depend on on which operations must be easy and efficient (for comparison of encoding schemes, see Koopman, Plasmeijer, & Jansen, 2014). The idea that learners must create entire frameworks for understanding, or *theories*, comes from cognitive and developmental literature emphasizing the way in which concepts and internal mental representations relate, to create systems of knowledge (Carey, 1985; Murphy & Medin, 1985; Wellman & Gelman, 1992; Gopnik & Meltzoff, 1997; Carey, 2009; Ullman et al., 2012). Of course, these relationships must include a variety of levels of abstraction—specific representations, computational processes, abstract rules, new symbols and concepts, etc. LCL permits this by providing a uniform language for all the components that might comprise a theory. If this aspect of LCL is correct, that might help to explain why cognitive science—and its theories of conceptual representation in particular—seem so hard to figure out. Definitions, prototypes, associations, or simple logical expressions seem like they would license fairly straightforward investigation by experimental psychology. But if concepts are intrinsically linked to others by conceptual and inferential role, then it may not be easy or possible to study much in controlled isolation.

LCL theories come from a simple basis. Turing machines are simple when compared to modern microprocessors but they are not simple when compared to combinatory logic. A Turing machine has separate mechanisms for its state, memory, and update rules. Combinatory logic has only a few functions that always perform the same operation on a binary branching tree. Indeed, the combinators **S&K** are not even minimal. There exist *single* combinators from which **S&K** can be derived. Single-point systems have been studied primarily as curiosities in logic or computer science, or as objective ways to measure complexity (Stay, 2005), but they suggest that the full complexity of human-like cognition may not be architecturally or genetically difficult to create. Thus, while it is difficult to see how to get the dynamics of a Turing machine into a neural network (though see Graves, Wayne, & Danihelka, 2014) it is easy get Turing-completeness through combinators (see Section below). The uniformity of representation—everything is built from just two elements—additionally means that the neural systems at the heart of the biological implementation potentially involve just a few types of neurons or local architectures. The implications for cognitive theories should be clear: a system that computes, even symbolically, need not be architecturally complex.

LCL theories are dynamical. A popular view is that cognitive systems are best viewed not as symbolic, but rather dynamical (Van Gelder, 1995, 1998; Beer, 2000). It's always a curious claim because by definition, everything in the universe is a dynamical system, even—or perhaps especially—Turing machines. LCL theories are inherently dynamical since the meaning of abstract symbols comes from the underlying dynamics of combinator evaluation, executed mindlessly by the underlying machine architecture. The view of dynamics emphasizes a key difference to traditional LOT theories. In most incarnations of the LOT the key part of having a concept would be building the structure (e.g. $CAUSE(x, GO(x, UP))$) and little attention is paid to the system by which this represents a computation that actually *runs*—what is the architecture of the evaluator, how does it run, and what makes those symbols mean what they do? In LCL theories, the important part of the representation is not only building the structure, but being able to run it or evaluate it with respect to other concepts. This, in some sense, puts the computational process itself back into "computational theory of mind"—we should not discuss symbols without discussing their computational/CRS roles.

For LCL, the important part of the dynamics is that it can be manipulated into capturing the relations present in any other system. Unlike most dynamical accounts in cognition, the dynamics are discrete in time and space; research on discrete space systems is a subfield of dynamical systems research in itself (Lind & Marcus, 1995) and is likely to hold many clues for how symbolic or symbolic-like processes may emerge out of underlying physics and biology. The general idea, for instance, of discretizing physical systems in order to provide adequate explanations lies at the heart of symbolic dynamics, including mathematical characterizations of general complex systems (Shalizi & Crutchfield, 2001). Recent work in cognitive science has explored the tradeoff between symbolic and continuous systems, providing a formal account of when systems may become discretized (Feldman, 2012).

LCL meanings are sub-symbolic and emergent. While LCL dynamics do deal with discrete objects (like combinators), the *meaning* of these objects is not inherent in the symbols themselves. Cognitive symbols like `True` arise from the *sub*-symbolic structures that `True` gets mapped to and the way these structures interact with other representations. While **S**&**K** can be defined symbolically, they can also be defined in terms of these underlying dynamics and interactions. This emphasis on sub-symbolic dynamics draws in part on connectionism, but also on theories that pre-date modern connectionism. Hofstadter (1985), for example, stresses the *active* nature of symbolic representations (also Hofstadter, 1980, 2008). He contrasts himself with Newell & Simon, for whom symbols are "just" the objects that get manipulated. For Hofstadter, symbols are the objects that participate in the manipulating:

> The brain itself does not "manipulate symbols"; the brain is the medium in which the symbols are floating and in which they trigger each other. There is no central manipulator, no central program. There is simply a vast collection of "teams"—patterns of neural firings that, like teams of ants, trigger other patterns of neural firings. The symbols are not "down there" at the level of the individual firings; they are "up here" where we do our verbalization. We feel those symbols churning within ourselves in somewhat the same way as we feel our stomach churning; we do not do symbol manipulation by some sort of act of will, let alone some set of logical rules of deduction. We cannot decide what we will next think of, nor how our thoughts will progress.
>
> Not only are we not symbol manipulators; in fact, quite to the contrary, we are manipulated by our symbols!

There are two separable claims here. One is that sub-symbols are active representational elements that conspire to give rise to symbols. This is shared by LCL theories. The other is that the substrate on which sub-symbols live is an unstructured medium in which symbols are "floating" and interact with each other without central control. This is a question of architecture and implementation which may be compatible with LCL in general (but not ChurIso's current implementation).

"Meaning" in this sense is *emergent* (McClelland et al., 2010) because it is not specified by any explicit feature of the design. Instead, the meaning emerges out of the LCL combinators' dynamics as well as the constraints (the data) that say which specific structures are most appropriate in a given domain. Emergentism from dynamics rather than architecture may capture why emergent phenomena can be found in many types of models, including the Bayesian ones (e.g Lee, 2010).

Recall that part of the motivation for LCL was that we wanted to formalize how symbols get meaning in order to better handle critiques like Searle's that "mere" symbol manipulation cannot explain the semantics of representation. As Chalmers (1992) argues, Searle's argument fails to apply to *sub*-symbolic systems like connectionist models because the computational (syntactic) elements are not intended to be meaningful themselves, in contrast to, e.g. (Newell & Simon, 1976). The same logic saves LCL from Searle's argument: **`S`**`&`**`K`** are not meant to, on their own, possess meaning other than dynamics, so the question of how meaning arises is answered only at a level higher than symbol manipulation.

LCL theories eagerly find patterns and use them in generalization. An important part of generalization is the ability to infer unobserved properties from those that can be observed, a task studied as *property induction* in psychology (Rips, 1975; Carey, 1985; Gelman & Markman, 1986; Osherson, Smith, Wilkie, Lopez, & Shafir, 1990; Shipley, 1993; Tenenbaum et al., 2006). In the **magnetism** example above, the attraction relations between an item `x` was fully and correctly inferred from only observing its interactions with a single other object. More generally, the ability to detect patterns may be a key feature of human thought, and many machine learning techniques center on extracting simple regularities from observed data. Table 7 shows two simple examples where ChurIso is provided with two properties, `dangerous` and `small` applied to to some objects (`a`, `b`, `x`, and `c`). Note that `small` doesn't have to be a feature per se, but could be a category membership predicate (e.g. `is-wolf`). In the first row, **property induction**, there is a perfect correlation between being dangerous and being small. In this case what we learn is a representation for these symbols where `(dangerous x)` $\rightarrow$ `True` even though all we know is that `x` is small. Being small, in fact, convinces the learner that `x` will have an identical conceptual role to `a`. This happens because in many cases, the easiest way for `x` to behave the correct way with respect to `small` is to make it the same as `a`. This illustrates a clear willingness to generalize, even from a small amount of data, a feature of human induction (Markman, 1991; Fei-Fei, Fergus, & Perona, 2006; Xu & Tenenbaum, 2007; Salakhutdinov, Tenenbaum, & Torralba, 2010; Lake, Salakhutdinov, & Tenenbaum, 2015). In the second case, the learner observes a third point, `c`, that is `small` but not `dangerous`. ChurIso decides that `small` things are still dangerous but that `c` is an exception, as shown by the fact that if all we know about `x` is that it is small, it still maps to the same combinator structure as `a`. This is a form of automatic creation of a simple form of a rule plus exception model, popular in categorization (Nosofsky, Palmeri, & McKinley, 1994; Nosofsky & Palmeri, 1998).

LCL theories supports deduction and simulation. The combinator structures that are learned are useful because they provide a way to derive new information. By combining operations in new ways (e.g. taking `(succ (succ (succ four)))`), learners are able to create the corresponding mental structures. This generative capacity is important in capturing the range of structures humans internalize. The ability can be viewed through two complementary lenses. The first is that LCL knowledge provides a deductive system which allows new knowledge to be proved. We can determine, for instance, whether

```
(succ (succ (succ three))) = (succ (succ four))
```

and thereby use our induced representations to learn about the world, since these representations are isomorphic to some structure in the world that matters. This view of knowledge is reminiscent of early AI attempts grounded in logic (see Nilsson, 2009) and cognitive theories of natural reasoning through deduction (Rips, 1989, 1994).

The second way to view the knowledge of an LCL system is as a means for mental simulation: one step forward of a combinator evaluation or one composition of two symbols corresponds to one step forward in a simulation of the relevant system. Simulation has received the most attention in physical understanding (e.g Hegarty, 2004; Battaglia et al., 2013) and folk psychology (e.g Gordon, 1986; Goldman, 2006), both of which are controversial (Stone & Davies, 1996; Davis & Marcus, 2016). However, the literature on simulation has focused on simulations of particular (e.g. physical) processes and not on LCL's goal of capturing arbitrary, relational or algorithmic aspects of the world. In general, simulation may be the primary evolutionary purpose of constructing a representation, as it permits use in novel situations, a phenomenon with clear behavioral benefits (Bubic, Von Cramon, & Schubotz, 2010).

LCL theories support learning. The mapping from symbols in base facts to combinators is solved by applying a bias for representational *simplicity* (Feldman, 2003b; Chater & Vitányi, 2003) and *speed* (Hutter, 2005; Schmidhuber, 2007). LCL theories inherit from the Bayesian LOT, and more generally work on program induction, a coherent computational-level theory of learning that provably works under a broad set of situations. Roughly, adopting the Bayesian setting, the data supports a hypothesis that most accurately matches the true generating distribution in the world. With enough data, then, learners will come to favor a hypothesis which is equivalent to the true one (see, e.g., Piantadosi, 2011). In this setting where we consider hypotheses to be programs or systems of knowledge written in a LCL system, with enough data ideal learners will be able to discover hypotheses that internalize their observed dynamics about the world.

As I have described above, a prior that favors hypotheses with short running times (e.g. $\exp(-\text{running time})$) permits learners to consider in principle hypotheses of arbitrary computational complexity. The trick is that the non-halting hypotheses have zero prior probability ($\exp(-\infty) = 0$) and this can be used to weed them out from a search or inference scheme that only needs upper-bounds on probability (like the Metropolis-Hastings algorithm). The ability to learn in such complex systems contrasts with arguments from the poverty of the stimulus in language learning and other areas of cognition. Note that this learning theory is stated as a computational theory, not an algorithmic one. There is still a question about how learners actually navigate the space of theories and hypotheses.

The ability to learn also provides an important consideration to conceptual theories (see Carey, 2015). When considering cognitive development as a computational process, it might be tempting to think that the simpler algorithmic components must be “built in” for learners, a perspective shared by many LOT learning setups. But it may be that developmental studies will not bear this out—it is unlikely, for instance, that very young children have any ability to handle arbitrary boolean logical structures. But if boolean logic is not innate, in what sense could it be learned? It seems far too simple a system to be implemented in a computational system *unless* the computational system has something equivalent to it to begin with. The LCL shows how it is possible: what is built-in may be a general system for constructing isomorphisms, and learners may have to realize the particular the structure () only when it is needed to explain data they observe. Due to the Turing-completeness of combinatory logic, this fact is true for all such representations learners might consider.

LCL theories' induction and generalization is sensitive to simplicity. In an LCL system without primitives, simplicity is determined through the built-in combinators. The set of primitives included determines what concepts are simple. For instance, if the Y combinator is included as a basic primitive, recursive functions may be expressible in just a few operations; if Y must be derived from **S** and **K**, then recursive operations will be much slower (requiring more evaluation steps) and more complex.

Work in computer science has suggested combinatory logic and related minimalistic computational formalisms should be used as the *definition* of complexity (Chaitin, 1982; Tromp, 2007). In the philosophical literature, Goodman (1983)'s *grue* problem points out that it is difficult to make simplicity fully objective. However, psychologically, there is a clear matter of fact as to which concepts are simple. In fact, the existence of a simplicity preference in concept learning provides a powerful tool: we can use people's psychological learning curves in order to reverse-engineer what formal system best captures their psychological measure of simplicity (for LOT work, see Kemp, 2012; Piantadosi et al., 2016). By applying similar methods, we might be able to discover which combinator basis generalizes most similarly to how people do.

Features of the implementation which are not part of the general LCL theory

Because we have been required to make some (as of yet) under-determined choices in order to implement an LCL theory, it is important to also describe which of these specific choices are not critical to the general theory I am advocating. The implementation I describe chooses *particular* combinators **S**&**K**, but there are infinitely many logically equivalent systems that could be empirically distinguished based on the inductive biases they imply. Because the brain is noisy and probabilistic, it is likely that probabilities interact with representations, perhaps even in a foundational way, as in recent work on Church (Goodman, Mansinghka, et al., 2008). These facts are intentionally ignored here in order to more cleanly articulate how the system may work more broadly. The primary claim is then that the right system likely shares many of the above features of combinatory logic.

It is also important to emphasize that many other formal systems have dynamics equivalent to LCL, some of which drop constraints such as compositionality or tree-structures. Cellular automata, for instance, rely only on local communication and computation, yet also are Turing-complete. If systems of cellular automata could be found that implement useful conceptual roles, they would qualify as a system very much like combinatory logic (although not in all the ways described above). Indeed, a lesson from complex systems research is that there are a huge variety of simple systems that are capable of universal computation (Wolfram, 2002), suggesting that it would not be hard, in principle, for nature to implement CRS-like representations.

A connectionist implementation

The LCL instantiation presented above lives most naturally at Marr (1982)'s representational and computational levels. However, it is important to understand how LCL theories may be implemented in a neural architecture. It turns out, this is relatively straightforward, in large part because the meaning of symbols arises from dynamics. We need two things: a means of representing tree structures (to represent a combinator composition) and a sequence of operations on those tree structures that implement **S**&**K**. Many ways of representing tree structures in neural dynamics have been considered (e.g. Pollack, 1990; Touretzky, 1990; Smolensky, 1990; Plate, 1995; Van Der Velde & De Kamps, 2006). Related work has also explored how vector spaces might encode compositional knowledge or logic (Rocktäschel, Bosnjak, Singh, & Riedel, 2014; Bowman, Potts, & Manning, 2014b, 2014a; Bowman, Manning, & Potts, 2015; Neelakantan, Roth, & McCallum, 2015; Gardner, Talukdar, & Mitchell, 2015; Smolensky et al., 2016).

Here, I will focus on Smolensky (1990)'s *tensor product* encoding (see Smolensky & Legendre, 2006). Section 3.7.2 of Smolensky (1990) showed how the Lisp operations `first` ↪ , `rest`, and `pair` could be implemented in a tensor product system. These are the only operations needed in order to implement an evaluator for combinatory logic (or something like it). Note that the idea of implementing these operations in a connectionist network is quite distinct from implementing them in **S**&**K** above. The above construction in **S**&**K** ↪ is meant to explain cognitive manipulation of tree structures as a component of high-level thought. The implementation in tensor products could be considered to be part of the neural hardware which is "built in" to human brains—the underlying dynamics from which symbols **S**&**K** themselves emerge. These innate processes could happen at the level of biological neurons, or at higher levels that are still consistent with the perspective of parallel, distributed processing.

To implement **S**&**K**, we can define a function called `(reduce t)` that computes the next step of the dynamics (the "→" operation):

```
(reduce t) := (first (rest t))                                    if
     (first t)  is  K
(reduce t) := (pair (pair (first t) (first (rest (rest t))))
                    (pair (first (rest t)) (first (rest (rest t)))))   if
                         (first t)  is  S
```

The left hand side of `(reduce t)` tell us that we are defining how the computational process of combinator evaluation may be carried out. The right hand side consists only of `first`, `rest`, and `pair` functions on binary trees, which, we assume, are here implemented in a connectionist network, following the methods of Smolensky (1990) or a similar approach. This function operates on data (combinators) that are themselves encoded with `pair`.

For instance, the structure `(K x y)` would be encoded in a connectionist network as `(pair K (pair x y))`. Then, following the definition of **reduce**,

```
(reduce (K x y)) = (reduce (pair K (pair x y)) → (first (rest (pair K (pair x y
   ↪ )))) → x
```

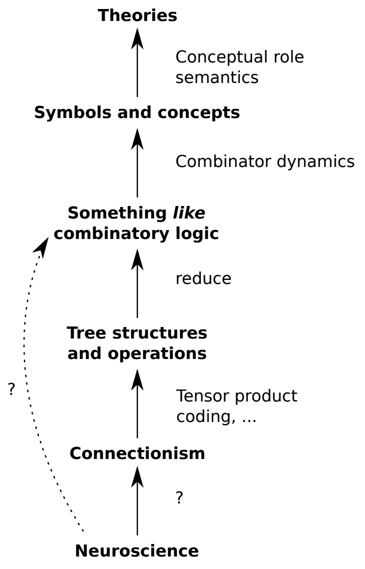

Theories
Conceptual role semantics
Symbols and concepts
Combinator dynamics
Something *like* combinatory logic
reduce
Tree structures and operations
Tensor product coding, ...
Connectionism
?
?
Neuroscience

Figure 2. An overview of the encoding of LCL dynamics into a connectionist architecture. Schemes like Smolensky (1990)'s tensor product encoding allow tree operations and structure (e.g. `first`, `rest`, `pair`), which can be used to implement the structures necessary for combinatory logic as well as the evaluator. The structures built in combinatory logic, as shown in this paper, create symbolic concepts which participate in theories and whose meaning is derived through conceptual role in those theories. It is possible that the intermediate levels below LCL are superfluous, and that dynamics *like* combinatory logic could be encoded directly in biological neurons (dotted line).

The case of **`S`** is analogous.

To summarize, Figure 2 shows a schematic of the general setup: tensor product coding (or an alternative) can be used to encode a tree structure in a connectionist network. The **`reduce`** function can then be stated as tensor operations, and these implement the dynamics of **`S`**&**`K`**, or a system like it. Then, combinator structures can be built with `pair` ↪ . The way these structures interact through **`reduce`** can give rise to create structured, algorithmic systems of knowledge through the appropriate Church encoding. In fact, any of the encoding schemes that supports `first`, `rest`, and `pair` can implement LCL theories, thereby permitting a swath of symbolic AI and cognitive science to be implemented in neural systems. For instance, abstract concepts like **recursion**, **repetition**, and **dominance** can be encoded directly using these methods into connectionist architectures. This, of course, treats connectionism as only an implementational theory of a CRS system. But this is, just a simplification for the present paper—there are certain to be areas where the parallel and distributed nature of connectionist theories are critically important (Rumelhart & McClelland, 1986), particularly at the interfaces of sensation and perception.

Conveniently, **`reduce`** is parallelizable. LCL inherits this advantage from functional programming languages. Taking terminology from computer science, parallel execution is possible because LCL representations are *referentially transparent*, meaning that a combinator reduces in the same way, regardless of where it appears in the tree. As a result, two reductions in the same tree can happen in either order (a theorem known as the *Church-Rosser theorem* (Church & Rosser, 1936)). A good implementation might do multiple reductions at the same time.

It is notable that the move towards unifying high-level symbolic thought with theories of implementation has been almost entirely one-sided. There are many connectionist approaches that try to explain—or explain away—symbolic thought. However, *almost no* work on the symbolic side has sought to push down towards representations that could more directly be implemented. Many symbolic modelers—myself included—have considered the problem of implementation to lie squarely in the neural modelers' domain: connectionist networks should strive to show where symbols could come from. However, if the view of this section is correct, the "hard" problem of implementation was solved in principle a quarter century ago and the main sticking point has actually been on the symbolic side of thinking more carefully about how symbols might get their meaning.

Remaining gaps to be filled

The LCL theory I have presented has intentionally focused on a minimal representational theory for high-level concepts. In doing so, it has neglected a number of important considerations.

LCL and the interfaces

Missing from the description of LCL theories is a formalization of how such abstract operations might interface to perception and action. To some proponents of LOT-like systems for representation, the way in which representations relate to the outside world is of central concern. Miller and Johnson-Laird (1976) write,

> A dictionary is a poor metaphor for a person's lexical knowledge. Dictionaries define words in terms of words. Such definitions, like some semantic theories, may provide plausible accounts of the intensional relations between words, but their circularity is vicious. There is no escape from the round of words. Language can apply to the nonlinguistic world, however, and this fact cannot be ignored by a theory of meaning. It is perhaps the single most important feature of language; a theory that overlooks it will provide simply a means of translating natural language into theoretical language. Although such a theory may be extremely useful as a device for formulating intensional relations, its ultimate value rests on a tacit appeal to its users' extensional intuitions.

The CRS itself has also been criticized for its circularity where meanings are defined in terms of each other (see Greenberg & Harman, 2005; Whiting, 2006). LCL embraces this circularity and shows that it is not inherently problematic to the construction of a working computational system (see also chapter 4 of Abelson & Sussman, 1996). For a theory of concepts, the circularity is even desirable because it prevents us from pushing the problem of meaning off into someone else's field—linguistic philosophy, neuroscience, robotics, etc.

To avoid the circularity, many of the theories of where such high-level constructs come from are based on repeated abstraction of sensory-motor systems, perhaps maintaining tight links between conception and perception (e.g. Barsalou, 1999, 2008, 2010; Sloutsky, 2010).

The challenge with this view is in understanding how concepts come to be distinct from perception, or more abstract, generalizing beyond immediate experience (Mahon, 2015), a goal with some recent computational progress (Yildirim & Jacobs, 2012). From the LCL point of view, the primary difficulty with theories closely tied to perception is that they do not engage with the computational richness of full human cognition—they do not explain how it is that we are able to carry out such a wide variety of computational processes and algorithms. A good example to consider might be *tangram* puzzles, where the goal is to construct a given shape (e.g. a giraffe) from simpler ones (e.g. triangles, squares, diamonds, etc.). Mental imagery and perception is certain to be involved in solving this task. However, high-level processes are also required—search algorithms that determine what arrangements to try next, memory of the geometry of the target shape, knowledge about what transformations are possible (Can a square be rotated to form a diamond? A triangle?). It is these high level algorithms and knowledge that the LCL aims to capture.

On the flip-side, the theory I have described does not engage with perception and action, nor which is sensitive to the type of content its actions manipulate. However, "two-factor" theories of CRS that more closely connect to perception and action have previously been proposed (Harman, 1987; Block, 1997), with tight connections to the debate about mental imagery. At the very least, the perceptual systems for shape must interface with high-level concepts–perhaps by speaking languages that are inter-translatable. In the same way, a computer has a single representation language for its central processor; the various subsystems —graphics cards and hard drives—must speak languages that are translatable with the central processor so that they can coordinate complex computation. Thinking a little more concretely, consider some base facts for a **square** concept,

```
(number-of-edges square) → four
(number-of-vertices square) → four
(angles square) → (pair rt-angle (pair rt-angle (pair rt-angle rt-angle)))
(rotate square 45) → diamond
(rotate diamond 45) → square
...
```

Even if this is the right conceptual role for square, it must *also* be the case that perception speaks this language. For instance, when `rotate` is called, we may rely on perceptual systems in order to execute the rotation so that `rotate` is not itself an operation in pure **S** ↪ &**K**. The key is that whatever dedicated hardware does do the rotation, it must send back symbols like `diamond` and `square` that are interpretable on a high level. To see this, consider the wide variety of computational processes that `square` can participate in and the other theories that it is related to. Here is one: Suppose I have a square-shaped ink stamp ■. I stamp it once, rotate the stamp by 45 degrees, ◆, stamp it again. If I do that forever, what shape will I have stamped out? What happens if I rotate it 90 degrees instead? The abstraction we readily handle is even more apparent in questions like, "If I rotate it 1 degree vs 2 degrees, which will give me a star with more points?" Likely, we can answer that question without forming a full mental image, relying instead on high-level properties of numbers, shapes, and their inter-relations. The situation can get even more complex: what happens if I rotate it $\sqrt{2}$ degrees each time? Solving this last problem in particular seems to require both an imagistic representation (to picture the stamping) as well as high-level logical reasoning about non-perceptual theories—in this case, the behavior of irrational numbers. Even simple questions about images interface with our systems of knowledge.

However, one feature of perception that is notably *un*like LCL is that small perceptual changes tend to correspond to small representational changes, a kind of principle of *continuity.*[9] For instance, the representation of a rotated symbol "*B*" is likely to be very similar to "B". However, the representations of LCL—and logical theory learning more generally (Ullman et al., 2012)—appear not to obey this principle. Changing one of the constraints or one data point might have huge consequences for the underlying representation. In the same way, a single data point might lead a human to revise an intuitive theory or a scientific theory. As far as I am aware, there are not yet good general solutions to this problem of finding logic-like representations that have good continuity properties, or indeed that interface well with perceptual systems.

Theories of innateness

Much of the LCL theory has been motivated by coming up with a logical formalism which involves just a few operations and contains no innate high-level content. However, the *core knowledge* theory from cognitive development holds that there are at least a few areas of conceptual development where innate representations should be preferred to logical minimalism (Spelke, 2003; Spelke & Kinzler, 2007; Carey, 2009). For instance, newborn infants might not be neutral statistical learners of an LCL variety, but might bring biases to attend to social cues or represent objects and salient properties. Or they might come with some built in representations in special domains like object perception or number. Other work has sought to discover the semantic primitives of thought that are shared across languages and cultures (Miller & Johnson-Laird, 1976; Brown & Wierzbicka, 1997), perhaps good candidates for a non-minimal LOT.

However, there is a strong methodological reason to fully explore the space of minimalist learning setups. It is important to understand what is *possible* to learn from data. This provides a useful formal setup for understanding empirical results. For instance, what might it mean if infants are poor at computing logical disjunction? Without a theory like LCL, it might not seem possible that something as basic as conjunction and disjunction could be learned. Second, in many domains, nativist theories have been over-eagerly accepted due to feeble theories of learning. A problematic example can be found in language acquisition, where early work like Gold's Theorem (Gold, 1967) purported to show that key aspects of language could not be learned and must therefore be innate. This gave rise to a rich formal theory of learning (e.g. Wexler & Culicover, 1983) but one which ultimately rested on unreasonable assumptions (Johnson, 2004). Informed by machine learning, probability theory, and computer science, it was later shown that learners could indeed identify all relevant aspects of language in a precise formal sense, without needing strong innate constraints (Chater & Vitányi, 2007). The lesson is also demonstrated by LCL here: learning is surprisingly powerful, and theories of what is likely to be innate must be informed by the most aggressive learning theories.

[9]A mathematical function, for instance mapping the world to a representation, is continuous if boundedly small changes in the input give rise to boundedly small changes in the output.

Extensions in logic

I have presented a very simplified view of combinatory logic. Here I will discuss a few important points where prior mathematical work has charted out a useful roadmap for future cognitive theories.

First, as I have described it, any structure built from `S&K` is evaluable. In the **roshambo** example, for instance, we could consider a structure like `(win win)`. To us, this is semantically meaningless, but it evaluates to `draw`. The problem of incoherent compositions could be solved in principle by use of a *type system* that defines and forbids nonsensical constructs (see Hindley & Seldin, 1986; Pierce, 2002). Each symbol would be associated with a "type" that function composition would have to respect. For instance, since `rock`, `paper`, and `scissors` would be one type that is allowed to operate on itself; when it does so, it produces an outcome (`win`, `lose`, or `draw`) of a different type that should not be treated as a function or argument (thus preventing constructions like `(win win)`). The corresponding learning theory would have to create new types in each situation, but learning could be greatly accelerated by use of types to constrain the space of possible representations.

The second limitation of combinatory logic is one of expressivity. Combinatory logic is Turing-complete only in the sense that it can represent any computable function on natural numbers (Turing, 1937). But there are well-defined operations that it cannot express, in particular about the nature of combinator structures themselves (see Theorem 3.1 of Jay & Given-Wilson, 2011). There is no combinator that can expression *equivalence* of two normal forms (Jay & Vergara, 2014). That is, we cannot construct a combinator `E` such that `(E x`
$\hookrightarrow$ `y)` reduces to `True` if and only if `x` and `y` are the same combinator structure. There are, however, formal systems that extend combinatory logic that can solve this problem (Curry & Feys, 1958; Wagner, 1969; Kearns, 1969; Goodman, 1972; Jay & Kesner, 2006; Jay & Given-Wilson, 2011), and these provide promising alternatives for future cognitive work. These types of logical systems will likely be important to consider, given the centrality of notions like same/different in cognitive processes, with hallmarks in animal cognition (Martinho & Kacelnik, 2016).

Finally, the semantics of combinatory logic is deterministic: evaluation always results in the same answer. However, probabilistic programming languages like Church (Goodman, Mansinghka, et al., 2008) provide inherently stochastic meanings that support conditioning and probabilistic inference, with nice consequences for theories of conceptual representation (Goodman et al., 2015), including an ability to capture gradience, uncertainty, and statistical inference. Combinatory logic could similarly be augmented with probabilistic semantics, and this may be necessary to capture the statistical aspects of human cognition (Tenenbaum et al., 2011).

The architecture itself

The connectionist implementation described above only shows how evaluation can be captured in a connectionist network. However, it does not provide an implementation of the entire architecture. A full implementation must include at least three other mechanisms.

First an implementation needs a means for observing base facts from the world and representing them. ChurIso assumes base facts are given as symbolic descriptions of the world. There are many possibilities that might inform these facts, including basic observation, pedagogical considerations, knowledge of similarity and features, or transfer and analogy across domains. The mind will likely have to work with these varied types of information in order to learn the correct theories. Extending the LCL theory beyond explicit relational facts is an important direction for moving the theory from the abstract into more psychological implementations. Similarly, a fuller theory will require understanding the level of granularity for base facts. Unless you are a particle physicist, you probably don't have detailed models of particle physics in your head. If we consider an LCL encoding of even an ordinary object, there are many possibilities: we might have an LCL structure for the whole object (e.g. the `rock` in **roshambo**); we might map its component parts to combinators which obey the right relations; we might map its edges and surfaces to combinator structures; or we might go below the level of surfaces. The question of granularity for LCL is simply an empirical question, and a goal for cognitive psychology should be to discover what level of description is correct.

Second, an important next step would be to understand the connectionist or neural implementation of search and inference. Likely this will be closely connected to a neural theory of probabilities and beliefs (Pouget, Beck, Ma, & Latham, 2013). For LCL, these beliefs must characterize learners' certainty that a given system of mapping symbols to combinators is the correct one.

Finally, the full architecture will require a means of processing symbols themselves. I have intentionally been somewhat vague about what it means to realize a symbol. There are several possibilities, but the choice between them will need to be informed by theories of the cognitive architecture. We might follow an "identity theory" for LCL where representing a symbol X is identical with activating its associated combinator structure. Thus, when we think NOT all that happens is we activate the structure `(S (S K K)(S K K))`. A challenge there is to understand how these structures also come to be associated with verbal labels, like a question of the form of memory. An alternative is that we have both symbols (as labels) and combinator structures (as meanings). In this case, NOT would be associated with its semantics `(S (S K K)(S K K))`. This fits most closely with the view above that a "variable" is a term that can be manipulated on its own.

What is the level of granularity?

My focus on isomorphism has also neglected a central question of what level of granularity is required for the isomorphism. Unless you are a particle physicist, you probably don't have detailed models of particle physics in your head. If we consider an LCL encoding of even an ordinary object, there are many possibilities: we might have an LCL structure for the whole object (e.g. the `rock` in **Roshambo**); we might map its component parts to combinators which obey the right relations; we might map its edges and surfaces to combinator structures; or we might go below the level of surfaces. The question of granularity for LCL is simply an empirical question, and a goal for cognitive psychology should be to discover what level of description is correct.

A related question is what level of granularity is required when computations take place. For instance, it easily could be the case that most reasoning, simulation, or prediction—all just combinator reduction—may work not through actual evaluation of combinators, but through memorized look-up tables of previous values, at least in part. Functional programming languages (including Goodman, Mansinghka, et al. (2008)'s Church) often include an ability to *memoize* a function, which means to remember the output of the function on each input rather than running it every time an input is provided. Understanding the relationship between re-derivation and memory (see, e.g. O'Donnell, 2015) is critical to formulating the psychological process.

General Discussion

One reason for considering LCL theories is that they provide a new riff on a variety of important issues, each of which I will discuss in turn.

Critiques of CRS

CRS is not without its critics (see Greenberg & Harman, 2005; Whiting, 2006). One argument from Fodor and Lepore (1992) holds that conceptual/inferential roles are not compositional, but meaning in language are. Therefore, meanings cannot be determined by their role. To illustrate, the meaning of "brown cow" is determined by "brown" and "cow". However, if we also know in our conceptual/inferential role that brown cows are dangerous (but white cows are not) then this is not captured compositionally. To translate this idea into LCL, imagine that composition is simply `pair`ing together `brown` and `cow` to form `(pair` ↪ `brown cow)`. How might our system know that the resulting structure is dangerous, if danger is not a part of the meanings (combinator structures) for `brown` or `cow`? A simple answer `(pair brown cow)` will have to interface to memory, and our remembered knowledge of the world will tell us so. One version of this is that the information is encoded in the meaning of either `is-dangerous` or `pair` (not `brown` and `cow`, as Fodor assumes). More concretely, the following base facts roughly capture this example:

```
(is-dangerous cow )  → False
(is-dangerous brown) → False
(is-dangerous (pair brown cow)) → True ; brown cows are dangerous
(is-brown brown) → True
(is-brown cow)   → False ; Cows are not generally brown
(is-brown (pair brown y))) → True ; anything brown is brown
```

Given these facts and the standard definitions of `True`, `False`, and `pair`, ChurIso finds

```
is-brown := ((S (S K K)) (S K))
cow      := ((S (S K K)) (K K))
brown    := (K (K K))
is-dangerous := ((S ((S S) K)) S)
```

This shows at a minimum that issues of compositionality and role are not so simple: when "role" is defined in a real system, compositionality can become subtle. Contrary to the misleadingly informal philosophical debate, consideration of a real system shows that it is in principle straightforward for a system to satisfy the required compositional roles.

Of course, it is unlikely that memory speaks logic in this way. The reason is that it would be very difficult to add new information since doing so would require changing the internals of a concept's representational structure. If suddenly we learned that we were mistaken and brown cows were *not* dangerous, we'd have to alter the meaning of one of the symbols rather than simply encode a new fact. Much better would be to have memory function as a look-up table, where combinator structures provide the index. Equivalently, `is-dangerous` might not be a combinator structure, but a special interface to a memory system (for an argument on the importance of memory architectures, see Gallistel & King, 2009). In this way, `is-dangerous` might be a fundamentally different kind of thing than a statement which is true *because* of the inherent properties of `brown` and `cow` (like a predicate `is-brown`). The argument therefore requires that we accept that there different kinds of statements—some of which are true in virtue of their meaning ("Brown cows are brown") and some of which are true in virtue of how the world happens to be ("Brown cows are dangerous"). Famously, Quine (1951) rejected the distinction between these kinds of properties, a view followed by Fodor and Lepore (1992) in their critique of CRS (see Block (1997) for a discussion of these issues in CRS and Fodor and Pylyshyn (2014) for more critiques). But to a psychologist, it's hard to see how a cognitive system that has both memory of the world and compositional meanings could be any other way; in fact, the mismatch between memory and compositional concepts is what drives us to learn and change conceptual representations themselves. Computers, too, certainly have some properties that can be derived from objects themselves *and* objects that can index facts in a database.

Fodor and Lepore (1992) also argue that CRS commits one *holism*, where meaning depends critically on all other aspects of knowledge, since these other aspects factor into conceptual role. Holism is considered harmful in part because it would be unclear how two people could hold the same beliefs or knowledge since it is unlikely that all components of their inferential system are the same. The difficulty with learning very complex systems of knowledge is also made clear in ChurIso: larger systems that have many symbols and relations tend to present a tougher constraint-satisfaction problem. One solution here is to favor modularity: in a real working computational system, the set of defining relations for a symbol might be small and circumspect. The logical operators, for instance, may only be defined with respect to each other, and not to combinator structures that represent an entirely different domain. Even a single object—for instance a representation of a **square**—could have different sets of combinators to interface with different operations (e.g. rotation vs. flipping). Such modularity of roles and functions is a desirable feature of computational systems in general, and the search to manage such complexity can be considered one of the defining goals of computer science (Abelson & Sussman, 1996).

A third challenge to CRS is in its handling of meaning and reference. There is a titanic literature on the role of reference in language and cognition, including work arguing for its centrality in conceptual representation (e.g. Fodor & Pylyshyn, 2014). To illustrate the importance of reference, Putnam (1975) considers a "twin earth" where there exists as substance that behaves in every way like water (H_2O) but is in fact composed of something else (XYZ). By assumption, XYZ plays the same role in conceptual systems as H_2O and yet is must be a different meaning since it refers to an entirely different substance. Any characterization of conceptual systems entirely by conceptual roles and relations will miss an important part of meaning. The problem leads others like Block (1997) discusses a *two-factor* theory of CRS in which concepts are identified by both their conceptual role and their reference (see also Harman, 1987). Critiques of the two-factor CRS are provided in Fodor and Lepore (1992) and discussed in Block (1997); a deflationary argument about the whole debate can be found in Gertler (2012). My inclination is that Putnam's argument tells us primarily about the meaning of the word "meaning" rather than anything substantive about the nature of mental representations (for a detailed cognitive view along these lines in a different setting, see Piantadosi, 2015). It is true that intuitively the meaning of a term should include something about its referent; it is not clear that our intuitions about this word tell us anything about how brains and minds actually work. Moreover, a difficult problem with theories based on reference is that there is no idea about what reference might mean computationally or scientifically—how we might tell if a system does or does not have the required referential properties.

The origin of novelty and conceptual change

A strange argument in philosophy of mind comes from Fodor (1975), who holds that there is an important sense in which most of our concepts must be innate. The argument goes, roughly, that the only way we learn new concepts is through composition of existing concepts. Thus, if we start with `GO` and `UP`, we can learn "lift" as `GO(x, UP(x))`. Fodor notes, however, that almost none of our concepts appear to have compositional formulations like these (see Margolis & Laurence, 1999). He concludes, therefore, that learning cannot create most of our concepts. The only possibility then is that almost all of our concepts are innate, including famously concepts as obscure as *carburetor*. While some of taken this as a reductio ad absurdum of the LOT or of compositional learning, it's hard to ignore the suspicion that Fodor's argument is simply mistaken in some way.

Indeed combinatory logic and other computational formalisms based on function composition show that it is: *any computational process can be expressed as a composition in a formal language or LOT* (see Piantadosi & Jacobs, 2016). The present paper shows that the LOT need not have *any* innate meanings—just innate dynamics. This means that if a computational theory of mind is correct—computations are the appropriate description for concepts like *carburetor*—then these must be expressible compositionally and therefore can be learned in Fodor's sense. A compositional CRS like LCL solves, at least in principle, the problem of explaining how an organism could learn so many different computations without requiring innate content on a cognitive level.

A corollary is that LCL theories can coherently formalize a notion of conceptual change, and that the processes of novel conceptual creation are inherently linked to the creation of systems of concepts, following (Laurence & Margolis, 2002; Block, 1987). One key question motivating this work is how learning could work if children lack knowledge of key computational domains like logic, number, or quantification. The idea that mental representations are like programs—a modern version of the LOT—has an implicit assumption that the primitives of these programs are the genetic endowment that humans are born with. We might be born with the ability to execute very elementary computations like `if` statements and logical disjunction, and put together those operations in order to express more complex cognitive content. However, this metaphor requires that infants—and newborns!—already have an ability to execute these abstract computations.

LCL shows one way in which this metaphor can be revised to include the possibility that such fundamental logical abilities are not innate, but built through experience. Learners could come with an ability to execute only underlying dynamical operations like **S**&**K**, thus possessing a system for potentially building theories and representations. In terms of *cognitive* content, this system would be a blank slate. Knowledge of **S**&**K** is not cognitive because they correspond to dynamical operations below the level of symbols, algorithms, and structures. Thus, theories of the world (e.g. systems of combinator structures) would be constructed in this universal system, rather than innately provided. To manage construction, learners would take perceptual observations (here, base facts), and use these to build internal models of the observed relationships between observed symbols. LCL therefore provides one concrete metaphor for what starting point could be cognitively minimal, yet still permit learners to discover arbitrarily rich systems of knowledge.

This provides a working theory at the interface of conceptual change and meaning. Learners' representations change when they successfully form an internal mental model of data they have observed, and the meaning of this new mental content would be specified entirely in its relationship to other cognitive domains (as well as the perceptual links it maintains, as discussed below). Concretely, a domain like **number** of **dominance** shows a case where learners would not have the requisite conception before internalizing the base facts. Once they map facts to a combinator structure, they will have discovered and created a fundamentally new conceptual object.

The origin of human-like cognition

LCL permits formalization of a variety of hypotheses about how humans' computational processes may differ qualitatively from other species. One purported defining feature of humans is our ability to make and use symbols, an idea with a long history throughout cognitive science (Deacon, 1997; Hurford, 2004), AI (Newell & Simon, 1976) and even the humanities (Burke, 1963). Recent work on the Bayesian LOT examines algorithms for the creations of new symbols, and the consequences thereof (Dechter, Malmaud, Adams, & Tenenbaum, 2013). It is easy to imagine that once symbols are defined in an LCL system, we are able to use those symbolic elements in new computations, potentially changing the inductive bias of the model. LCL also raises a distinct possibility in this spirit: human-level intelligence may arise from the ability to associate *multiple* conceptual roles with a single symbol. For instance, in the **existential** example, it is much easier to use a different combinator for `True` than in the **Boolean** example. It would be possible to require them to have the same combinator structure, but doing so would lead to representations that are much less concise and efficient. It may be that the ability to associate multiple distinct roles with a single symbol is what permits easy internalization of such a variety of computational processes. This problem may also address Fodor's "brown cow" problem, in that the composition "brown cow" may come to be associated with additional roles beyond just "brown" and "cow."

Alternatively—or perhaps complementarily—theories that hypothesize **recursion** is central (Hauser et al., 2002; Corballis, 2014) would suppose that the `Y`-combinator is memorized, high-probability, or innately encoded in humans. More generally, there may be specific combinators that humans have—arising from language or other pressures—that give our cognition its distinctive properties.

A related possibility is that human cognition differs in its ability to have some structures *without* types. In strongly-typed systems (see Chapters 10-14 of Hindley & Seldin, 1986), combinators lead all computations to halt. This raises the possibility that humans developed the ability to construct systems of arbitrary computational complexity by allowing ourselves to construct computational structures that do not terminate.

We can also consider the influence of expanding the bounds of, for instance, working memory (size of structures permitted) or the number of computational steps that can be executed. Each of these would drastically affect the computational power of the system. This fact means that human-like computational abilities might come from alterations in multiple cognitive systems (Premack, 2004), particularly those supporting the construction, maintenance, and evaluation of LCL structures.

A final hypothesis is that human cognition is inherently tied to mental logics which support thinking about their own internal representations and processes. Full programming languages like Scheme and LISP (Smith, 1984; Abelson & Sussman, 1996) support reasoning about representations already with a `quote` operation that converts a program expression into data and `eval` that converts data into a program. Lambda calculi (Wand, 1998; Stump, 2009) and combinatory logic (Jay & Given-Wilson, 2011) have been developed that directly support meta-computation.

Conclusion: Towards a synthesis

Cognitive science enjoys an embarrassment of riches. There are many seemingly incompatible approaches to understanding cognitive processes and no consensus view on which is right or even how they relate to each other. The major debates seem to be in large part disagreements of which *metaphor* is right for thinking about the brain. These debates have made two things clear: none of our metaphors are yet sufficient, and none of them is completely misguided. Cognitive systems are dynamical systems; they give rise to structured manipulation of symbols; they also are implemented in physical/biological systems whose dynamics are determined far below the level of mental algorithms. Our learning supports inference of broad classes of computations, yet clearly we have something built in that differs from other animals. The LCL can be thought of as a new metaphor—a sub-meaningful symbolic-dynamical system that gives rise straightforwardly to the types of structures, representations, and algorithms that permeate cognitive science, *and* that is implementable directly in neural architectures. In spanning these levels it avoids dodging questions about meaning.

If anything like the resulting theory is correct, there are important consequences for theories of conceptual change as the LOT. Conceptual change from a strikingly minimal basis to arbitrarily systems of knowledge is possible if learners come with built-in dynamical objects and learning mechanisms. Theories of learning need not assume any cognitive content in principle, not even the basics of familiar computational systems, like logic and number. The key is in formalizing a theory of the meaning of mental content; if CRS is chosen, it permits construction of these systems of knowledge from much less. The framework I have described follows the LOT in positing a structured, internal language for mentalese. But it differs from most instantiations of the LOT in that the primitives of the language aren't cognitive at all. **`S&K`** formalize the underlying neural (sub-cognitive) dynamics and it is only in virtue how structures built of these dynamics interact that meaningful systems of thought can arise. Thus, the idea of a formal language for thinking was right; the idea that the language has primitives with intrinsic meaning—beyond their dynamics—was not.

A trope in cognitive science is that we need more constraints in order to narrow the space of possible theories. Each subfield chooses its own constraints—architectural, rational, neural, computational etc. One thing that I hope to highlight with LCL is that the idea of wanting more constraints—e.g. narrowing in on a single kind of hypothesis—

might be misguided or premature. Additional constraints are useful when the pool of theories is too large and must be culled. But it might be the case that we have too *few* theories in circulation, in that none of our approaches satisfactorily handle all that we know about cognitive processes. In this case, our search might benefit from expanding its set of metaphors—fewer constraints—to consider new kinds of formal systems as possible cognitive theories. LCL is just one attempt, with clear strengths and weaknesses, some of which I have highlighted above.

Perhaps the greatest strength of the resulting framework is that it unifies a variety of ideas in cognition. None of the formal machinery used here is original to this paper—all of it comes allied fields. The motivating ideas then conspire to create a theory that is extraordinarily simple: a few elementary operations on trees are composed productively, giving rise to a huge variety of possible cognitive structures and operations. Meanings are defined by the way these structures interact under the elementary operations' dynamics. This metaphor promising foundation on which to build a theory of mental representation, with a clear road map for further progress. This paper made assumptions to show how a system could actually work, but the general theory is not about any particular logical system, representational formalism, or cognitive architecture. Instead, I have tried to present arguments that combinatory logic captures a few general properties of thought, suggesting that mental representations, whatever they happen to be, will be *like* combinatory logic in a number of important ways.

Domain	Facts	Representation

Seasons	(succ winter) → spring
	(succ spring) → summer
	(succ summer) → fall
	(succ fall) → winter
		spring := (K (K K))
		winter := (K (S (K K) (K (K K)))
		↪)
		fall := (K K)
		summer := K
		succ := ((S ((S S) S)) K)

1, 2, Many	(succ one) → two
	(succ two) → many
	(succ many) → many
		many := (S K K)
		two := (K (S K K))
		one := K
		succ := ((S (S K K)) (K (S K K))
		↪)

Roshambo	(rock scissors) → win
	(rock rock) → draw
	(scissors scissors) →
	↪ draw
	(scissors paper) → win
	(paper rock) → win
	(paper paper) → draw
	(rock paper) → lose
	(paper scissors) → lose
	(scissors rock) → lose
		lose := (K K)
		paper := ((S ((S K) S)) (K (K K)
		↪))
		draw := K
		win := (K (K K))
		scissors := ((S ((S S) (K (K K))
		↪)) K)
		rock := ((S ((S S) (K (K (K K)))
		↪)) S)

Family tree	(father sasha) → barak
	(father malia) → barak
	(mother sasha) →
	↪ michelle
	(mother malia) →
	↪ michelle
	(sister malia) → sasha
	(sister sasha) → malia
	(husband michelle) →
	↪ barak
	(wife barak) →
	↪ michelle
		michelle := (K S)
		mother := (K (K S))
		malia := (K K)
		barak := S
		sasha := K
		father := (K S)
		husband := ((S (S K K)) S)
		sister := ((S (S K K)) (S K K))

Table 1
Church encoding inferred from the base facts that permit representation of several logical structures common in psychology. The arrow "↪" shows wrapped lines.

Domain	Facts	Representation

Singular/Plural	(marker item) →
	↪ singular
	(marker item item) →
	↪ plural
	(marker item item item)
	↪ → plural
	(marker item item item
	↪ item) → plural
		item := (**S** (**S** **K**))
		marker := (**S** (**K** (**S** **K** **K**)))
		plural := ((**S** (**S** **K**)) (**S** (**S** **K**)))
		singular := ((**S** (**K** (**S** **K** **K**))) (**S**
		↪ (**S** **K**)))

Number
($\mathbb{Z}$)
	(succ one) → two
	(succ two) → three
	(succ three) → four
		succ := **K**
		one := **S**
		two := (**K** **S**)
		three := (**K** (**K** **S**))
		four := (**K** (**K** (**K** **S**)))

Even-Odd
($\mathbb{Z}_2$)
	True := (**K** **K**)
	
	(succ one) → two
	(succ two) → three
	(succ three) → four
	
	(even one) ↛ True
	(even two) → True
	(even three) ↛ True
	(even four) → True
	
	(odd one) → True
	(odd two) ↛ True
	(odd three) → True
	(odd four) ↛ True
		odd := ((**S** ((**S** (**K** **S**) **K**) **K**)) (**S** **K**
		↪ **K**))
		even := ((**S** (**S** **K** **K**)) (**K** **K**))
		
		one := (((**S** (**K** (**S** (**K** (**S** **S** (**K** **K**))
		↪) **K**)) **S**) (**S** **K** **K**)) (**K** **K**))
		succ := (((**S** (**K** (**S** (**K** (**S** **S** (**K** **K**)
		↪)) **K**)) **S**) (**S** **K** **K**)) **K**)

Table 2
***S&K** structures in domains involving interesting generalization, where the combinator structures allow deduction beyond the base facts.*

Domain	Facts	Representation

Dominance
$(a \succ b \succ c \succ d)$
	`True := K`
	`(dom a b) → True`
	`(dom a c) → True`
	`; No information a,d`
	`↪ relation`
	`(dom b c) → True`
	`(dom b d) → True`
	`(dom c d) → True`
	`(dom b a) ↛ True`
	`(dom c a) ↛ True`
	`(dom c b) ↛ True`
	`(dom d b) ↛ True`
	`(dom d c) ↛ True`
	`(dom b a) ↛ True`
	`(dom c b) ↛ True`
	`(dom d c) ↛ True`
		`a := (K (K K))`
		`b := (S (S K))`
		`c := (S K K)`
		`d := (K K)`
		`dom := (((S (K (S (K (S S (K K))`
		`↪ ) K)) S) (S (K (S (K (S S`
		`↪ (K K))) K)) S)) K)`

Magnetism	`(attract p1 p2) ↛ True`
	`(attract p2 p1) ↛ True`
	`(attract p1 n1) → True`
	`(attract p1 n2) → True`
	`(attract p2 n1) → True`
	`(attract p2 n2) → True`
	`(attract n1 n2) ↛ True`
	`(attract n2 n1) ↛ True`
	`(attract n1 p1) → True`
	`(attract n1 p2) → True`
	`(attract n2 p1) → True`
	`(attract n2 p2) → True`
	`; and one single example`
	`(attract n1 x) → True`
	`True := (K K) ; fixed by`
	`↪ design`
		`attract := ((S S) (K I))`
		`n1 := K`
		`n2 := K`
		`p2 := (K K)`
		`p1 := (K K)`
		`x := (K K)`

Table 3
S&K *structures in domains involving interesting generalization, where the combinator structures allow deduction beyond the base facts.*

Domain	Facts	Representation
Reversal	`(reverse x y) → (y x)`	`reverse := ((S (K (S (S K K)))) ↪ K)`
If-else	`True := (K K)` `False := K` `(ifelse True x y) → x` `(ifelse False x y) → y`	`ifelse := ((S ((S K) S)) (S K)))`
Identity	`(identity x) → x`	`identity := (S K K)`
Repetition	`(repeat f x) → (f (f x) ↪ )`	`repeat := ((S (S (K S) K)) (S K ↪ K))`
Recursion	`(Y f) --→ (f (Y f))`	`Y := (((S (K S) K) ((S ((S (K (S ↪ (K (S S (K K))) K)) S) ( ↪ S (S (S K K))))) S)) (S ( ↪ K S) K))`
Mutual recursion	`(f (g (Y⋆ f g))) --→ (Y⋆ ↪ f g)`	`Y⋆ := (((S (K S) K) (((S (K S) K ↪ ) ((S (S (K S) K)) ((S (K ↪ (S (K (S S (K K))) K)) S ↪ ) (S (K S) K)))) (S ((S ( ↪ K (S (K (S S (K K))) K)) ↪ S) (S K K))))) (S (K S) K ↪ ))`
Apply	`(apply f x ) → (f x)`	`apply = (S K K)`
Tree, List	`(first (pair x y)) → x` `(rest (pair x y)) → y`	`pair := (((S (K S) K) (S (K (S ( ↪ K (S S (K K))) K)) S)) (( ↪ S (K (S (K (S S (K K))) K ↪ )) S) (S (K (S (K (S S (K ↪ K))) K)) S)))` `first := ((S (S K K)) (K (S K)) ↪ )` `rest := ((S (S K K)) (K K))`

Table 4
***S&K** structures that implement computational operations.*

Language	Facts	Representation

Regular
$((ab)^n)$
	(a start) → state_a
	(b state_a) → accept
	
	(a accept) → state_a
	(b accept) → invalid
	
	(a invalid) → invalid
	(b invalid) → invalid
		start := (**K** (**K** (**S** **K** **K**)))
		a := ((**S** (**S** **K** **K**)) **K**)
		b := ((**S** (**S** **K** **K**)) ((**S** (**K** (**S** (**K** (
		↪ **S** **S** (**K** **K**))) **K**)) **S**) (**S** **K** **K**
		↪)))
		
		invalid := (((**S** (**K** (**S** (**K** (**S** **S** (**K**
		↪ **K**))) **K**)) **S**) (**S** **K** **K**)) (**S**
		↪ **K** **K**))
		accept := (**S** **K** **K**)

Context-free
(a^nb^n)
	(a start) → got_a
	(b got_a) → accept
	
	(a got_a) → got_aa
	(b got_aa) → want_b
	(b want_b) → accept
	
	(a got_aa) → got_aaa
	(b got_aaa) → want_bb
	(b want_bb) → want_b
	
	(a got_aaa) → got_aaaa
	(b got_aaaa) → want_bbb
	(b want_bbb) → want_bb
		start := **S**
		a := (**S** (**K** **K**))
		b := (((**S** (**K** (**S** (**K** (**S** **S** (**K** **K**)))
		↪ **K**)) **S**) (**S** **K** **K**)) **S**)
		accept := (**K** (**S** **S**))

Existential
$(\exists z \dots)$
	(start True) → accept
	(start False) → reject
	
	(reject True) → accept
	(reject False) → reject
	
	(accept True) → accept
	(accept False) → accept
		True := ((**S** **S**) **K**)
		False := ((**S** (**S** **K** **K**)) **K**)
		start := ((**S** (**S** **K** **K**)) **K**)
		accept := ((**S** (**K** (**S** **S** **K**))) (**K** (**K**
		↪ (**S** **S** **K**))))
		reject := ((**S** (**S** **K** **K**)) **K**)

Finite {a, man, am, an, mam}	$\mathbb{S}$:= ↪ {__a, man, _am, _an, mam} For all s in $\{\_, m, a, n\}^3$ (check s) → valid if ↪ $s \in \mathbb{S}$ (check s) $\nrightarrow$ valid ↪ otherwise	_ = (((**S** (**K** (**S** (**K** (**S** **S** (**K** **K**))) **K** ↪)) **S**) (**S** (**K** (**S** (**K** (**S** **S** (**K** ↪ **K**))) **K**)) **S**)) (**S** (**K** (**S** (**K** ↪ (**S** **S** (**K** **K**))) **K**)) **S**)) n = (**K** ((**S** (**K** (**S** (**K** (**S** **S** (**K** **K**))) ↪ **K**)) **S**) (**S** (**K** (**S** (**K** (**S** **S** ↪ (**K** **K**))) **K**)) **S**))) a = (((**S** (**K** (**S** (**K** (**S** **S** (**K** **K**))) **K** ↪)) **S**) (**S** (**K** **S**) **K**)) **S**) m = (((**S** (**K** (**S** (**K** (**S** **S** (**K** **K**))) **K** ↪)) **S**) (**S** **K** **K**)) (**S** (**K** (**S** (↪ **K** (**S** **S** (**K** **K**))) **K**)) **S**)) valid = (((**S** (**K** (**S** (**K** (**S** **S** (**K** **K**) ↪)) **K**)) **S**) (**S** (**K** (**S** (**K** (**S** ↪ **S** (**K** **K**))) **K**)) **S**)) ((**S** (**K** ↪ (**S** (**K** (**S** **S** (**K** **K**))) **K**)) **S**) ↪ (**S** (**K** (**S** (**K** (**S** **S** (**K** **K**))) ↪ **K**)) **S**))) check = ((**S** (**S** (**S** (**K** (**S** (**K** (**S** **S** ↪ (**K** **K**))) **K**)) **S**))) (**S** (**K** **S**) ↪ **K**))

Table 5
Church encoding of several formal language constructs.

Function	Equivalent combinatory logic structure
`(f) → (+ 1 4)`	`f := (+ 1 4)`
`(f x) → (+ x 1)`	`f := (S + (K 1))`
`(f x y) → (+ x y)`	`f := +`
`(f op x y) → (op x y)`	`f := (S K K)`

Table 6
*Several levels of abstraction for a function `f` and their corresponding combinator structures. The combinator structures have no explicit variables (e.g. x, y, op). Note that if the constants or primitives `+`, `1`, and `4` were defined with a Church encoding, they too would be combinators, permitting us to translate everything into pure **S**&**K**.*

Domain	Facts	Representation

Property Induction	`True  = (small a)`
	`False = (small b)`
	`True  = (dangerous a)`
	`False = (dangerous b)`
	
	`True = (small x)`
		`small := (S K K)`
		`dangerous := ((S K) S)`
		
		`a := (K K)`
		`b := K`
		`x := (K K)`

Exception Induction	`True  = (small a)`
	`False = (small b)`
	`True  = (small c)`
	`True  = (dangerous a)`
	`False = (dangerous b)`
	`False = (dangerous c)`
	
	`True = (small x)`
		`small := ((S (S K K)) (S K))`
		`dangerous := ((S (S K K)) (S K K`
		`    ↪ ))`
		
		`a := (K (K K))`
		`b := (K K)`
		`c := (((S (K (S (K (S S (K K)))`
		`    ↪ K)) S) (S K K)) (K K))`
		`x := (K (K K))`

Table 7
Learners who observe a correlation between dangerousness and size will generalize based on size (top). Learners who see conflicted data count `c` as an exception.

Acknowledgements

I am extremely grateful to Goker Erdogan and Hayley Clatterbuck for providing detailed comments and suggesting improvements on an earlier draft of this work. Josh Rule contributed greatly to this work by providing detailed comments on an early draft, important discussion, and important improvements to ChurIso's implementation. Noah Goodman, Ralf Haefner, Josh Tenenbaum, Chris Bates, Matt Overlan, and members of the computation and language lab and kidd lab provided useful discussions relevant to these ideas.

Research reported in this publication was supported by the Eunice Kennedy Shriver National Institute of Child Health & Human Development of the National Institutes of Health under award number R01HD085996-01. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health.

Appendix
Sketch of universality

It may or may not be obvious to readers that any statement about the relation between objects can be encoded into an LCL system. Here, I sketch a simple proof that this is possible when we are allowed to define what function composition means. My focus is on the high-level logic of the proof while attempting to minimize the amount of notation required. Let's suppose that we are given an arbitrary base fact like,

```
(a (b x)) → (c (d e f))
```

We may re-write this into binary constraints, with a single variable on the left and a single function application on the right, by introducing "dummy" variables `D1`, `D2`, etc:

```
(d e)  → D1 ; right hand term is D1-D3
(D1 f) → D2
(c D2) → D3 ; D3 enforces the equality between the sides
(b x)  → D4 ; left hand term is D3-D4
(a D4) → D3
```

This is akin to Chomsky normal form for a context-free grammar.

The challenge then is to find a mapping from symbols to combinators that satisfies these expressions. A difficulty to note is that some variables, like `D1`, may appear on the left *and* the right, meaning that their combinator structure must be the output of a function (appearing on the left) as well as a function that itself does something useful (on the right). To address this, the proof sketch here will assume that we are allowed to define the way functions are applied. For instance, instead of requiring `(d e) → D1`, we will replace the function application `(d e)` with our own custom one, `(evluate d e)`. When `evaluate = ↪ I`, we are left with ordinary function application. I do not determine here if requiring `evaluate=I` permits universal isomorphism (I suspect not). But we can show that if we are free to choose `evaluate`, we can satisfy any constraints.

With this change, we can re-write our base facts as,

```
(evaluate d e) → D1 ; right hand term is D1-D3
(evaluate D1 f)→ D2
(evaluate c D2)  → D3 ; D3 enforces the equality between the sides
(evaluate b x)  → D4 ; left hand term is D3-D4
(evaluate a D4)→ D3
```

With this addition, we can take each of the symbols (`a b c d e f x` and `D1 D2 D3 D4`) and give them each an integer with Church encoding. Standard schemes for this can be found in Pierce (2002). Integers in Church encoding also support addition, subtraction, and multiplication. We may therefore view these facts as a set of integer-values, where `evaluate` is a function from two (integer) arguments to a single (integer) outcome:

```
(evaluate 4 5) → 8 ; (evaluate d e)
(evaluate 8 6) → 9 ; (evaluate D1 f)
(evaluate 3 9) → 10 ; (evaluate c D2)
(evaluate 2 7) → 11 ; (evaluate b x)
(evaluate 1 11) → 10; (evaluate a D4)
```

Note that at this point we may check if the facts are logically consistent—they may not state, for instance, that `(f x y)` → `z`, `(f x y)` → `w`, and `z` ≠ `w`.

Assuming consistency, we may then explicitly encode the facts by setting `evaluate` to be a polynomial which encodes these facts. To see how this is possible, suppose we have constraints

```
(evaluate α₁ β₁) → γ₁
(evaluate α₂ β₂) → γ₂
(evaluate α₃ β₃) → γ₃
...
```

It is well-known that in one dimension, any set of x, y points can be approximated by a polynomial. The same holds for two dimensions, with a variety of available techniques. This means that we can set `evaluate` to be the combinator that implements the polynomial mapping each α_i, β_i to γ_i with the desired accuracy.

An alternative to 2D polynomials is to use Gödel numbering to convert the two-dimensional problem to a one-dimensional one. If `evaluate` first converts its arguments to a single integer, for instance $2^{\alpha_i}3^{\beta_i}$, then the problem of finding the right polynomial reduces to a one-dimensional interpolation problem. Explicit solutions then exist, such as this version of Lagrange's solution to the general problem,

$$(\texttt{evaluate}\ \alpha_i\ \ \beta_i) := \sum_{j=1}^{n} \gamma_j \prod_{\substack{1\leq m\leq k\\ m\neq j}} \frac{2^{\alpha_i}3^{\beta_i} - 2^{\alpha_m}3^{\beta_m}}{2^{\alpha_j}3^{\beta_j} - 2^{\alpha_m}3^{\beta_m}}. \tag{2}$$

To check this, note that when $i = j$, the fractions inside the product cancel and the coefficient for γ_j becomes 1. However, when $i \neq j$, then there will be some numerator term which is zero, canceling out all of the other γ_m. Together, these give the output of `evaluate` as γ_i when given α_i and γ_i as input.

Note that this construction does not guarantee sensible generalizations when running `evaluate` on new symbols. The specific patterns of generalization will depend on how symbols are mapped to integers, but more problematically, polynomial interpolation famously exhibits chaotic or wild behavior on points other than those that are fixed, a fact known as *Runge's phenomenon* (Runge, 1901). As a result, the polynomial mapping should be taken only as an existence proof that some mapping of combinators will be able to satisfy the base facts, or the combinatory logic can in principle encode any isomorphism when we define function application with `evaluate`.

References

Abe, H., & Lee, D. (2011). Distributed coding of actual and hypothetical outcomes in the orbital and dorsolateral prefrontal cortex. *Neuron*, *70*(4), 731–741.

Abelson, H., & Sussman, G. (1996). *Structure and interpretation of computer programs.* Cambridge, MA: MIT Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological review*, *111*(4), 1036.

Anderson, J. R., Matessa, M., & Lebiere, C. (1997). Act-r: A theory of higher level cognition and its relation to visual attention. *Human-Computer Interaction*, *12*(4), 439–462.

Aydede, M. (1997). Language of thought: The connectionist contribution. *Minds and Machines*, *7*(1), 57–101.

Barsalou, L. W. (1999). Perceptions of perceptual symbols. *Behavioral and brain sciences*, *22*(04), 637–660.

Barsalou, L. W. (2008). Grounded cognition. *Annu. Rev. Psychol.*, *59*, 617–645.

Barsalou, L. W. (2010). Grounded cognition: Past, present, and future. *Topics in cognitive science*, *2*(4), 716–724.

Battaglia, P. W., Hamrick, J. B., & Tenenbaum, J. B. (2013). Simulation as an engine of physical scene understanding. *Proceedings of the National Academy of Sciences*, *110*(45), 18327–18332.

Beer, R. D. (2000). Dynamical approaches to cognitive science. *Trends in cognitive sciences*, *4*(3), 91–99.

Bennett, C. H. (1995). Logical depth and physical complexity. *The Universal Turing Machine A Half-Century Survey*, 207–235.

Blackburn, P., & Bos, J. (2005). *Representation and inference for natural language: A first course in computational semantics.* Center for the Study of Language and Information.

Block, N. (1987). Advertisement for a semantics for psychology. *Midwest studies in philosophy*, *10*(1), 615–678.

Block, N. (1997). Semantics, conceptual role. *The Routledge Encylopedia of Philosophy.*

Bonawitz, E. B., Schijndel, T. J. van, Friel, D., & Schulz, L. (2012). Children balance theories and evidence in exploration, explanation, and learning. *Cognitive psychology*, *64*(4), 215–234.

Boole, G. (1854). *An investigation of the laws of thought: on which are founded the mathematical theories of logic and probabilities.* London, UK: Walton and Maberly.

Bowman, S. R., Manning, C. D., & Potts, C. (2015). Tree-structured composition in neural networks without tree-structured architectures. *arXiv preprint arXiv:1506.04834*.

Bowman, S. R., Potts, C., & Manning, C. D. (2014a). Learning distributed word representations for natural logic reasoning. *arXiv preprint arXiv:1410.4176*.

Bowman, S. R., Potts, C., & Manning, C. D. (2014b). Recursive neural networks can learn logical semantics. *arXiv preprint arXiv:1406.1827*.

Brigandt, I. (2004). Conceptual role semantics, the theory theory, and conceptual change.

Brown, C. H., & Wierzbicka, A. (1997). *Semantics: Primes and universals.* JSTOR.

Bubic, A., Von Cramon, D. Y., & Schubotz, R. I. (2010). Prediction, cognition and the brain. *Frontiers in human neuroscience*, *4*, 25.

Burke, K. (1963). Definition of man. *The Hudson Review*, *16*(4), 491–514.

Cardone, F., & Hindley, J. R. (2006). History of lambda-calculus and combinatory logic. *Handbook of the History of Logic*, *5*, 723–817.

Carey, S. (1985). Conceptual change in childhood.

Carey, S. (2009). *The Origin of Concepts.* Oxford: Oxford University Press.

Carey, S. (2015). Why theories of concepts should not ignore the problem of acquisition. *Disputatio: International Journal of Philosophy*, *7*(41).

Chaitin, G. J. (1982). *Algorithmic information theory.* Wiley Online Library.

Chalmers, D. (1990). Why fodor and pylyshyn were wrong: The simplest refutation. In *Proceedings of the twelfth annual conference of the cognitive science society, cambridge, mass* (pp. 340–347).

Chalmers, D. J. (1992). Subsymbolic computation and the chinese room. *The symbolic and connectionist paradigms: Closing the gap*, 25–48.

Chater, N., & Oaksford, M. (1990). Autonomy, implementation and cognitive architecture: A reply to fodor and pylyshyn. *Cognition*, *34*(1), 93–107.

Chater, N., & Vitányi, P. (2003). Simplicity: A unifying principle in cognitive science? *Trends in Cognitive Sciences*, *7*(1), 19–22.

Chater, N., & Vitányi, P. (2007). Ideal learning of natural language: Positive results about learning from positive evidence. *Journal of Mathematical Psychology*, *51*(3), 135–163.

Chomsky, N. (1956). Three models for the description of language. *Information Theory, IRE Transactions on*, *2*(3), 113–124.

Chomsky, N. (1957). *Syntactic Structures.* The Hague: Mouton.

Church, A. (1936). An unsolvable problem of elementary number theory. *American journal of mathematics*, *58*(2), 345–363.

Church, A., & Rosser, J. B. (1936). Some properties of conversion. *Transactions of the American Mathematical Society*, *39*(3), 472–482.

Conant, R. C., & Ross Ashby, W. (1970). Every good regulator of a system must be a model of that systemâĂă. *International journal of systems science*, *1*(2), 89–97.

Corballis, M. C. (2014). *The recursive mind: The origins of human language, thought, and civilization.* Princeton University Press.

Craik, K. J. W. (1967). *The nature of explanation.* CUP Archive.

Curry, H. B., & Feys, R. (1958). *Combinatory logic, volume i of studies in logic and the foundations of mathematics.* North-Holland Amsterdam.

Davies, D., & Isard, S. D. (1972). Utterances as programs. *Machine intelligence*, *7*, 325–339.

Davis, E. (1990). Representations of commonsense.

Davis, E., & Marcus, G. (2016). The scope and limits of simulation in automated reasoning. *Artificial Intelligence*, *233*, 60–72.

Davis, M., & Putnam, H. (1960). A computing procedure for quantification theory. *Journal of the ACM (JACM)*, *7*(3), 201–215.

Deacon, T. (1997). *The symbolic species.* New York: Norton.

Dechter, E., Malmaud, J., Adams, R. P., & Tenenbaum, J. B. (2013). Bootstrap learning via modular concept discovery. In *Proceedings of the twenty-third international joint conference on artificial intelligence* (pp. 1302–1309).

Drews, C. (1993). The concept and definition of dominance in animal behaviour. *Behaviour*, *125*(3), 283–313.

Ebbinghaus, H.-D., & Flum, J. (2005). *Finite model theory.* Springer Science & Business Media.

Ediger, B. (2011). *cl - a combinatory logic interpreter.* http://www.stratigery.com/cl/.

Erdogan, G., Yildirim, I., & Jacobs, R. A. (2015). From sensory signals to modality-independent conceptual representations: A probabilistic language of thought approach. *PLoS Comput Biol*, *11*(11), e1004610.

Falkenhainer, B., Forbus, K. D., & Gentner, D. (1986). *The structure-mapping engine.* Department of Computer Science, University of Illinois at Urbana-Champaign.

Fei-Fei, L., Fergus, R., & Perona, P. (2006). One-shot learning of object categories. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *28*(4), 594–611.

Feldman, J. (2000). Minimization of Boolean complexity in human concept learning. *Nature*, *407*(6804), 630–633.

Feldman, J. (2003a). Simplicity and complexity in human concept learning. *The General Psychologist*, *38*(1), 9–15.

Feldman, J. (2003b). The simplicity principle in human concept learning. *Current Directions in Psychological Science*, *12*(6), 227.

Feldman, J. (2012). Symbolic representation of probabilistic worlds. *Cognition*, *123*(1), 61–83.

Field, H. H. (1977). Logic, meaning, and conceptual role. *The Journal of Philosophy*, *74*(7), 379–409.

Florêncio, C. (2002). Learning generalized quantifiers. In *Proceedings of Seventh ESSLLI Student Session.*

Fodor, J. (1975). *The language of thought.* Cambridge, MA: Harvard University Press.

Fodor, J. (1997). Connectionism and the problem of systematicity (continued): Why smolensky's solution still doesn't work. *Cognition*, *62*(1), 109–119.

Fodor, J. (2008). *LOT 2: The language of thought revisited.* Oxford: Oxford University Press.

Fodor, J., & Lepore, E. (1992). Holism: A shopper's guide.

Fodor, J., & McLaughlin, B. P. (1990). Connectionism and the problem of systematicity: Why Smolensky's solution doesn't work. *Cognition*, *35*(2), 183–204.

Fodor, J., & Pylyshyn, Z. (1988). Connectionism and cognitive architecture: a critical analysis, Connections and symbols. *A Cognition Special Issue, S. Pinker and J. Mehler (eds.)*, 3–71.

Fodor, J., & Pylyshyn, Z. W. (2014). *Minds without meanings: An essay on the content of concepts.* MIT Press.

Frege, G. (1892). Über sinn und bedeutung. *Wittgenstein Studien*, *1*(1).

French, R. M. (2002). The computational modeling of analogy-making. *Trends in cognitive Sciences*, *6*(5), 200–205.

Gallistel, C., & King, A. (2009). *Memory and the computational brain.* New York: Wiley Blackwell.

Gallistel, C. R. (1998). Symbolic processes in the brain: The case of insect navigation. *An invitation to cognitive science*, *4*, 1–51.

Gardner, M., Talukdar, P., & Mitchell, T. (2015). Combining vector space embeddings with symbolic logical inference over open-domain text. In *2015 aaai spring symposium series* (Vol. 6, p. 1).

Gayler, R. W. (2004). Vector symbolic architectures answer jackendoff's challenges for cognitive neuroscience. *arXiv preprint cs/0412059*.

Gayler, R. W. (2006). Vector symbolic architectures are a viable alternative for jackendoff's challenges. *Behavioral and Brain Sciences*, *29*(01), 78–79.

Gelman, S. A., & Markman, E. M. (1986). Categories and induction in young children. *Cognition*, *23*(3), 183–209.

Gentner, D. (1983). Structure-mapping: A theoretical framework for analogy. *Cognitive science*, *7*(2), 155–170.

Gentner, D., & Markman, A. B. (1997). Structure mapping in analogy and similarity. *American psychologist*, *52*(1), 45.

Gentner, D., & Stevens, A. L. (1983). Mental models.

Gertler, B. (2012). Understanding the internalism-externalism debate: What is the boundary of the thinker? *Philosophical Perspectives*, *26*(1), 51–75.

Gierasimczuk, N. (2007). The problem of learning the semantics of quantifiers. *Logic, Language, and Computation*, 117–126.

Gold, E. (1967). Language identification in the limit. *Information and control*, *10*(5), 447–474.

Goldman, A. I. (2006). *Simulating minds: The philosophy, psychology, and neuroscience of mindreading.* Oxford University Press.

Goodman, N. (1983). *Fact, fiction, and forecast.* Harvard University Press.

Goodman, N., Mansinghka, V., Roy, D., Bonawitz, K., & Tenenbaum, J. (2008). Church: A language for generative models. In *Proceedings of the 24th conference on uncertainty in artificial intelligence, uai 2008* (pp. 220–229).

Goodman, N., Tenenbaum, J., Feldman, J., & Griffiths, T. (2008). A Rational Analysis of Rule-Based Concept Learning. *Cognitive Science*, *32*(1), 108–154.

Goodman, N. D. (1972). A simplification of combinatory logic. *The Journal of Symbolic Logic*, *37*(02), 225–246.

Goodman, N. D., Tenenbaum, J. B., & Gerstenberg, T. (2015). Concepts in a probabilistic language of thought. In Margolis & Lawrence (Eds.), *The conceptual mind: New directions in the study of concepts.* MIT Press.

Goodman, N. D., Ullman, T. D., & Tenenbaum, J. B. (2011). Learning a theory of causality. *Psychological review*, *118*(1), 110.

Gopnik, A., & Meltzoff, A. N. (1997). *Words, thoughts, and theories.* Mit Press.

Gopnik, A., & Wellman, H. M. (2012). Reconstructing constructivism: Causal models, bayesian learning mechanisms, and the theory theory. *Psychological bulletin*, *138*(6), 1085.

Gordon, R. M. (1986). Folk psychology as simulation. *Mind & Language*, *1*(2), 158–171.

Graves, A., Wayne, G., & Danihelka, I. (2014). Neural turing machines. *arXiv preprint arXiv:1410.5401*.

Greenberg, M., & Harman, G. (2005). Conceptual role semantics.

Griffiths, T., Chater, N., Kemp, C., Perfors, A., & Tenenbaum, J. (2010). Probabilistic models of cognition: exploring representations and inductive biases. *Trends Cogn. Sci*, *14*(10.1016).

Grosenick, L., Clement, T. S., & Fernald, R. D. (2007). Fish can infer social rank by observation alone. *Nature*, *445*(7126), 429–432.

Grünwald, P. D. (2007). *The minimum description length principle.* MIT press.

Hadley, R. F. (2009). The problem of rapid variable creation. *Neural computation*, *21*(2), 510–532.

Harman, G. (1987). (Non-solipsistic) Conceptual Role Semantics. In E. Lepore (Ed.), *New directions in semantics.* London: Academic Press.

Hauser, M. D., Chomsky, N., & Fitch, W. T. (2002). The faculty of language: What is it, who has it, and how did it evolve? *science*, *298*(5598), 1569–1579.

Hegarty, M. (2004). Mechanical reasoning by mental simulation. *Trends in cognitive sciences*, *8*(6), 280–285.

Heim, I., & Kratzer, A. (1998). *Semantics in generative grammar.* Malden, MA: Wiley-Blackwell.

Hindley, J., & Seldin, J. (1986). *Introduction to combinators and λ-calculus.* Cambridge, UK: Press Syndicate of the University of Cambridge.

Hoffman, D. D., Singh, M., & Prakash, C. (2015). The interface theory of perception. *Psychonomic bulletin & review*, *22*(6), 1480–1506.

Hofstadter, D. R. (1980). Gödel Escher Bach. *New Society*.

Hofstadter, D. R. (1985). Waking up from the boolean dream. *Metamagical Themas*, 631–665.

Hofstadter, D. R. (2008). *I am a strange loop.* Basic books.

Hopcroft, J., Motwani, R., & Ullman, J. (1979). *Introduction to automata theory, languages, and computation* (Vol. 3). Addison-wesley Reading, MA.

Hsu, A., & Chater, N. (2010). The logical problem of language acquisition: A probabilistic perspective. *Cognitive science*, *34*(6), 972–1016.

Hsu, A., Chater, N., & Vitányi, P. (2011). The probabilistic analysis of language acquisition: Theoretical, computational, and experimental analysis. *Cognition*, *120*(3), 380–390.

Hurford, J. R. (2004). Human uniqueness, learned symbols and recursive thought. *European Review*, *12*(04), 551–565.

Hutter, M. (2005). *Universal artificial intelligence: Sequential decisions based on algorithmic probability.* Springer Science & Business Media.

Jackendoff, R. (2002). *Foundation of language-brain, meaning, grammar, evolution.* Oxford: Oxford university press.

Jacobson, P. (1999). Towards a variable-free semantics. *Linguistics and philosophy*, *22*(2), 117–185.

Jay, B., & Given-Wilson, T. (2011). A combinatory account of internal structure. *The Journal of Symbolic Logic*, *76*(03), 807–826.

Jay, B., & Kesner, D. (2006). Pure pattern calculus. In *Programming languages and systems* (pp. 100–114). Springer.

Jay, B., & Vergara, J. (2014). Confusion in the church-turing thesis. *arXiv preprint arXiv:1410.7103*.

Johnson, K. (2004). Gold's theorem and cognitive science. *Philosophy of Science*, *71*(4), 571–592.

Johnson-Laird, P. N. (1977). Procedural semantics. *Cognition*, *5*(3), 189–214.

Johnson-Laird, P. N. (1983). *Mental models: Towards a cognitive science of language, inference, and consciousness* (No. 6). Harvard University Press.

Katz, Y., Goodman, N., Kersting, K., Kemp, C., & Tenenbaum, J. (2008). Modeling semantic cognition as logical dimensionality reduction. In *Proceedings of Thirtieth Annual Meeting of the Cognitive Science Society.*

Kearns, J. T. (1969). Combinatory logic with discriminators. *The Journal of symbolic logic*, *34*(4), 561–575.

Kemp, C. (2012). Exploring the conceptual universe. *Psychological Review*, *119*, 685–722.

Kemp, C., & Tenenbaum, J. (2008). The discovery of structural form. *Proceedings of the National Academy of Sciences*, *105*(31), 10687.

Kemp, C., Tenenbaum, J. B., Griffiths, T. L., Yamada, T., & Ueda, N. (2006). Learning systems of concepts with an infinite relational model. In *Aaai* (Vol. 3, p. 5).

Kemp, C., Tenenbaum, J. B., Niyogi, S., & Griffiths, T. L. (2010). A probabilistic model of theory formation. *Cognition*, *114*(2), 165–196.

Koopman, P., Plasmeijer, R., & Jansen, J. M. (2014). Church encoding of data types considered harmful for implementations.

Kwiatkowski, T., Goldwater, S., Zettlemoyer, L., & Steedman, M. (2012). A probabilistic model of syntactic and semantic acquisition from child-directed utterances and their meanings. In *Proceedings of the 13th conference of the european chapter of the association for computational linguistics* (pp. 234–244).

Kwiatkowski, T., Zettlemoyer, L., Goldwater, S., & Steedman, M. (2010). Inducing probabilistic ccg grammars from logical form with higher-order unification. In *Proceedings of the 2010 conference on empirical methods in natural language processing* (pp. 1223–1233).

Lake, B. M., Salakhutdinov, R., & Tenenbaum, J. B. (2015). Human-level concept learning through probabilistic program induction. *Science*, *350*(6266), 1332–1338.

Laurence, S., & Margolis, E. (2002). Radical concept nativism. *Cognition*, *86*(1), 25–55.

Lee, M. D. (2010). Emergent and structured cognition in bayesian models: comment on griffiths et al. and mcclelland et al. *Update*, *14*(8).

Liang, P., Jordan, M., & Klein, D. (2010). Learning Programs: A Hierarchical Bayesian Approach. In *Proceedings of the 27th International Conference on Machine Learning.*

Libkin, L. (2013). *Elements of finite model theory.* Springer Science & Business Media.

Lind, D., & Marcus, B. (1995). *An introduction to symbolic dynamics and coding.* Cambridge University Press.

Loar, B. (1982). Conceptual role and truth-conditions: comments on harman's paper:"conceptual role semantics". *Notre Dame Journal of Formal Logic*, *23*(3), 272–283.

Mahon, B. Z. (2015). What is embodied about cognition? *Language, cognition and neuroscience*, *30*(4), 420–429.

Marcus, G. F. (2003). *The algebraic mind: Integrating connectionism and cognitive science.* MIT press.

Margolis, E., & Laurence, S. (1999). *Concepts: core readings.* The MIT Press.

Markman, E. M. (1991). *Categorization and naming in children: Problems of induction.* Mit Press.

Marr, D. (1982). *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information.* W.H. Freeman & Company.

Martinho, A., & Kacelnik, A. (2016). Ducklings imprint on the relational concept of âĂIJsame or differentâĂİ. *Science*, *353*(6296), 286–288.

McClelland, J. L., Botvinick, M. M., Noelle, D. C., Plaut, D. C., Rogers, T. T., Seidenberg, M. S., et al. (2010). Letting structure emerge: connectionist and dynamical systems approaches to cognition. *Trends in cognitive sciences*, *14*(8), 348–356.

Miller, G. A., & Johnson-Laird, P. N. (1976). *Language and perception.* Belknap Press.

Mollica, F., & Piantadosi, S. T. (2015). Towards semantically rich and recursive word learning models. *Proceedings of the Cognitive Science Society.* Retrieved from `http://colala.bcs.rochester.edu/papers/mollica2015towards.pdf`

Mostowski, M. (1998). Computational semantics for monadic quantifiers. *Journal of Applied Nonclassical Logics*, *8*, 107–122.

Murphy, G. L., & Medin, D. L. (1985). The role of theories in conceptual coherence. *Psychological review*, *92*(3), 289.

Neelakantan, A., Roth, B., & McCallum, A. (2015). Compositional vector space models for knowledge base inference. In *2015 aaai spring symposium series.*

Newell, A. (1994). *Unified theories of cognition.* Cambridge, MA: Harvard University Press.

Newell, A., & Simon, H. A. (1976). Computer science as empirical inquiry: Symbols and search. *Communications of the ACM*, *19*(3), 113–126.

Nieuwenhuis, R., Oliveras, A., & Tinelli, C. (2006). Solving sat and sat modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll (t). *Journal of the ACM (JACM)*, *53*(6), 937–977.

Nilsson, N. J. (2009). *The quest for artificial intelligence.* Cambridge University Press.

Nosofsky, R., Palmeri, T., & McKinley, S. (1994). Rule-plus-exception model of classification learning. *Psychological review*, *101*(1), 53.

Nosofsky, R. M., & Palmeri, T. J. (1998). A rule-plus-exception model for classifying objects in continuous-dimension spaces. *Psychonomic Bulletin & Review*, *5*(3), 345–369.

O'Donnell, T. J. (2015). *Productivity and reuse in language: A theory of linguistic computation and storage.* MIT Press.

Okasaki, C. (1999). *Purely functional data structures.* Cambridge University Press.

Osherson, D. N., Smith, E. E., Wilkie, O., Lopez, A., & Shafir, E. (1990). Category-based induction. *Psychological review*, *97*(2), 185.

Overlan, M. C., Jacobs, R. A., & Piantadosi, S. T. (2016). A hierarchical probabilistic language-of-thought model of human visual concept learning. *Proceedings of the Cognitive Science Society.* Retrieved from `http://colala.bcs.rochester.edu/papers/overlan2016hierarchical.pdf`

Penn, D. C., Holyoak, K. J., & Povinelli, D. J. (2008). Darwin's mistake: Explaining the discontinuity between human and nonhuman minds. *Behavioral and Brain Sciences*, *31*(02), 109–130.

Piantadosi, S. T. (2011). *Learning and the language of thought.* Unpublished doctoral dissertation, MIT. Retrieved from `http://colala.bcs.rochester.edu/papers/piantadosi_thesis.pdf`

Piantadosi, S. T. (2015). Problems in the philosophy of mathematics: A view from cognitive science. In E. Davis & P. J. Davis (Eds.), *Mathematics, substance and surmise: Views on the meaning and ontology of mathematics.* Springer. Retrieved from http://colala.bcs.rochester.edu/papers/piantadosi2015problems.pdf

Piantadosi, S. T., Goodman, N., & Tenenbaum, J. (under review). Modeling the acquisition of quantifier semantics: a case study in function word learnability. Retrieved from http://colala.bcs.rochester.edu/papers/piantadosi2012modeling.pdf

Piantadosi, S. T., & Jacobs, R. (2016). Four problems solved by the probabilistic Language of Thought. *Current Directions in Psychological Science*, *25*, 54–59. Retrieved from http://colala.bcs.rochester.edu/papers/piantadosi2016four.pdf

Piantadosi, S. T., Tenenbaum, J., & Goodman, N. (2012). Bootstrapping in a language of thought: a formal model of numerical concept learning. *Cognition*, *123*, 199–217. Retrieved from http://colala.bcs.rochester.edu/papers/piantadosi2012bootstrapping.pdf

Piantadosi, S. T., Tenenbaum, J., & Goodman, N. (2016). The logical primitives of thought: Empirical foundations for compositional cognitive models. *Psychological Review.* Retrieved from http://colala.bcs.rochester.edu/papers/piantadosi2016logical.pdf

Pierce, B. C. (2002). *Types and programming languages.* MIT press.

Plate, T. A. (1995). Holographic reduced representations. *Neural networks, IEEE transactions on*, *6*(3), 623–641.

Pollack, J. B. (1990). Recursive distributed representations. *Artificial Intelligence*, *46*(1), 77–105.

Pouget, A., Beck, J. M., Ma, W. J., & Latham, P. E. (2013). Probabilistic brains: knowns and unknowns. *Nature neuroscience*, *16*(9), 1170–1178.

Premack, D. (2004). Is language the key to human intelligence? *Science*, *303*(5656), 318–320.

Putnam, H. (1975). The meaing of meaning. In *Philosophical Papers, Volume II: Mind, Language, and Reality.* Cambridge: Cambridge University Press.

Quine, W. V. (1951). Main trends in recent philosophy: Two dogmas of empiricism. *The philosophical review*, 20–43.

Rips, L., Asmuth, J., & Bloomfield, A. (2006). Giving the boot to the bootstrap: How not to learn the natural numbers. *Cognition*, *101*, 51–60.

Rips, L., Asmuth, J., & Bloomfield, A. (2008). Do children learn the integers by induction? *Cognition*, *106*, 940–951.

Rips, L., Asmuth, J., & Bloomfield, A. (2013). Can statistical learning bootstrap the integers? *Cognition*, *128*(3), 320–330.

Rips, L., Bloomfield, A., & Asmuth, J. (2008). From numerical concepts to concepts of number. *Behavioral and Brain Sciences*, *31*, 623–642.

Rips, L. J. (1975). Inductive judgments about natural categories. *Journal of verbal learning and verbal behavior*, *14*(6), 665–681.

Rips, L. J. (1989). The psychology of knights and knaves. *Cognition*, *31*(2), 85–116.

Rips, L. J. (1994). *The psychology of proof: Deductive reasoning in human thinking.* Mit Press.

Rocktäschel, T., Bosnjak, M., Singh, S., & Riedel, S. (2014). Low-dimensional embeddings of logic. In *Acl workshop on semantic parsing.*

Rogers, T., & McClelland, J. (2004). *Semantic cognition: A parallel distributed processing approach.* Cambridge, MA: MIT Press.

Rumelhart, D., & McClelland, J. (1986). *Parallel distributed processing.* Cambridge, MA: MIT Press.

Runge, C. (1901). Über empirische funktionen und die interpolation zwischen äquidistanten ordinaten. *Zeitschrift für Mathematik und Physik*, *46*(224-243), 20.

Salakhutdinov, R., Tenenbaum, J., & Torralba, A. (2010). One-shot learning with a hierarchical nonparametric bayesian model.

Schmidhuber, J. (2007). Gödel machines: Fully self-referential optimal universal self-improvers. In *Artificial general intelligence* (pp. 199–226). Springer.

Scholten, D. (2010). A primer for conant and ashbyâĂŹs good-regulator theorem. *Unpublished.*

Searle, J. R. (1980). Minds, brains, and programs. *Behavioral and brain sciences*, *3*(03), 417–424.

Sellars, W. (1963). Science, perception, and reality.

Shalizi, C. R., & Crutchfield, J. P. (2001). Computational mechanics: Pattern and prediction, structure and simplicity. *Journal of statistical physics*, *104*(3-4), 817–879.

Shastri, L., Ajjanagadde, V., Bonatti, L., & Lange, T. (1996). From simple associations to systematic reasoning: A connectionist representation of rules, variables, and dynamic bindings using temporal synchrony. *Behavioral and Brain Sciences*, *19*(2), 326–337.

Shepard, R. N., & Chipman, S. (1970). Second-order isomorphism of internal representations: Shapes of states. *Cognitive psychology*, *1*(1), 1–17.

Shipley, E. F. (1993). Categories, hierarchies, and induction. *The psychology of learning and motivation*, *30*, 265–301.

Siskind, J. (1996). A Computational Study of Cross-Situational Techniques for Learning Word-to-Meaning Mappings. *Cognition*, *61*, 31–91.

Sloutsky, V. M. (2010). From perceptual categories to concepts: What develops? *Cognitive science*, *34*(7), 1244–1286.

Smith, B. C. (1984). Reflection and semantics in lisp. In *Proceedings of the 11th acm sigact-sigplan symposium on principles of programming languages* (pp. 23–35).

Smolensky, P. (1988). The constituent structure of connectionist mental states: A reply to fodor and pylyshyn. *The Southern Journal of Philosophy*, *26*(S1), 137–161.

Smolensky, P. (1989). Connectionism and constituent structure. *Connectionism in perspective*, 3–24.

Smolensky, P. (1990). Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial intelligence*, *46*(1), 159–216.

Smolensky, P., Lee, M., He, X., Yih, W.-t., Gao, J., & Deng, L. (2016). Basic reasoning with tensor product representations. *arXiv preprint arXiv:1601.02745*.

Smolensky, P., & Legendre, G. (2006). *The Harmonic Mind.* Cambridge, MA: MIT Press.

Spelke, E. (2003). What makes us smart? Core knowledge and natural language. In

D. Gentner & S. Goldin-Meadow (Eds.), *Language in Mind.* Cambridge, MA: MIT Press.

Spelke, E., & Kinzler, K. (2007). Core knowledge. *Developmental Science*, *10*(1), 89–96.

Stay, M. (2005). Very simple chaitin machines for concrete ait. *Fundamenta Informaticae*, *68*(3), 231–247.

Steedman, M. (2001). *The syntactic process.* Cambridge MA: MIT Press.

Stone, T., & Davies, M. (1996). The mental simulation debate: A progress report. *Theories of theories of mind*, 119–137.

Stump, A. (2009). Directly reflective meta-programming. *Higher-Order and Symbolic Computation*, *22*(2), 115–144.

Tabor, W., Juliano, C., & Tanenhaus, M. K. (1997). Parsing in a dynamical system: An attractor-based account of the interaction of lexical and structural constraints in sentence processing. *Language and Cognitive Processes*, *12*(2-3), 211–271.

Tenenbaum, J., Kemp, C., Griffiths, T., & Goodman, N. (2011). How to grow a mind: Statistics, structure, and abstraction. *science*, *331*(6022), 1279–1285.

Tenenbaum, J. B., Griffiths, T. L., & Kemp, C. (2006). Theory-based bayesian models of inductive learning and reasoning. *Trends in cognitive sciences*, *10*(7), 309–318.

Tenenbaum, J. B., Kemp, C., & Shafto, P. (2007). Theory-based bayesian models of inductive reasoning. *Inductive reasoning: Experimental, developmental, and computational approaches*, 167–204.

Tiede, H. (1999). Identifiability in the limit of context-free generalized quantifiers. *Journal of Language and Computation*, *1*(1), 93–102.

Touretzky, D. S. (1990). Boltzcons: Dynamic symbol structures in a connectionist network. *Artificial Intelligence*, *46*(1), 5–46.

Tromp, J. (2007). Binary lambda calculus and combinatory logic. *Randomness and Complexity, from Leibniz to Chaitin*, 237–260.

Turing, A. M. (1937). Computability and λ-definability. *The Journal of Symbolic Logic*, *2*(04), 153–163.

Ullman, T., Goodman, N., & Tenenbaum, J. (2012). Theory learning as stochastic search in the language of thought. *Cognitive Development.*

van Benthem, J. (1984). Semantic automata. In J. Groenendijk, D. d. Jongh, & M. Stokhof (Eds.), *Studies in discourse representation theory and the theory of generalized quantifiers.* Dordrecht, The Netherlands: Foris Publications Holland.

Van Der Velde, F., & De Kamps, M. (2006). Neural blackboard architectures of combinatorial structures in cognition. *Behavioral and Brain Sciences*, *29*(01), 37–70.

Van Gelder, T. (1990). Compositionality: A connectionist variation on a classical theme. *Cognitive Science*, *14*(3), 355–384.

Van Gelder, T. (1995). What might cognition be, if not computation? *The Journal of Philosophy*, *92*(7), 345–381.

Van Gelder, T. (1998). The dynamical hypothesis in cognitive science. *Behavioral and brain sciences*, *21*(05), 615–628.

Wagner, E. G. (1969). Uniformly reflexive structures: on the nature of gödelizations and relative computability. *Transactions of the American Mathematical Society*, *144*, 1–41.

Wand, M. (1998). The theory of Fexprs is trivial. *Lisp and Symbolic Computation*, *10*(3), 189–199.

Wellman, H. M., & Gelman, S. A. (1992). Cognitive development: Foundational theories of core domains. *Annual review of psychology*, *43*(1), 337–375.

Wexler, K., & Culicover, P. (1983). *Formal principles of language acquisition.* Cambridge, MA: MIT Press.

Whiting, D. (2006). Conceptual role semantics.

Wisniewski, E. J., & Medin, D. L. (1994). On the interaction of theory and data in concept learning. *Cognitive Science*, *18*(2), 221–281.

Wolfram, S. (2002). *A new kind of science* (Vol. 1). Wolfram Media Champaign, IL.

Wong, Y. W., & Mooney, R. J. (2007). Learning synchronous grammars for semantic parsing with lambda calculus. In *Annual meeting-association for computational linguistics* (Vol. 45, p. 960).

Woods, W. A. (1968). Procedural semantics for a question-answering machine. In *Proceedings of the december 9-11, 1968, fall joint computer conference, part i* (pp. 457–471).

Woods, W. A. (1981). *Procedural semantics as a theory of meaning.* (Tech. Rep.). DTIC Document.

Xu, F., & Tenenbaum, J. (2007). Word learning as Bayesian inference. *Psychological Review*, *114*(2), 245–272.

Yildirim, I., & Jacobs, R. A. (2012). A rational analysis of the acquisition of multisensory representations. *Cognitive science*, *36*(2), 305–332.

Yildirim, I., & Jacobs, R. A. (2013). Transfer of object category knowledge across visual and haptic modalities: Experimental and computational studies. *Cognition*, *126*(2), 135–148.

Yildirim, I., & Jacobs, R. A. (2014). Learning multisensory representations for auditory-visual transfer of sequence category knowledge: a probabilistic language of thought approach. *Psychonomic bulletin & review*, 1–14.

Zettlemoyer, L. S., & Collins, M. (2005). Learning to Map Sentences to Logical Form: Structured Classification with Probabilistic Categorial Grammars. In *UAI* (pp. 658–666).