Theories of Mind as Languages of Thought for Thought about Thought

Kartik Chandra and **Jonathan Ragan-Kelley** and **Joshua B. Tenenbaum**
MIT CSAIL and Brain & Cognitive Sciences

Abstract

What kind of thing is a theory of mind? How can we formalize the various theories of mind we study as cognitive scientists? In this paper, we argue that it is valuable to think of a theory of mind as a kind of programming language: one that is specialized for setting up and reasoning about problems involving other minds. Drawing on ideas from the theory and history of programming languages, we show how this perspective can help us formalize concepts in a theory of mind, precisely articulate differences between multiple theories of mind, and reason about how we develop our theories-of-mind over time.

Introduction

How should we think about a theory of mind? On the influential "theory-theory" (Carey, 1985; Gopnik & Meltzoff, 1997), we should think of a theory-of-mind as, indeed, a kind of theory: akin to scientific theories (Gopnik & Wellman, 1992). But what exactly do we mean by a "theory," here? How can we articulate a given theory-of-mind θ in precise, computational terms? And what kind of thing is θ, anyway—how can we formalize the space $\mathcal{T}$ of possible alternate theories-of-mind that $\theta \in \mathcal{T}$ is a member of?

A precise mathematical language for representing theories-of-mind is valuable for three key reasons. First, such a language would allow us to state our ideas and hypotheses about theories-of-mind in explicit, unambiguous terms. Second, such a language would allow us to systematically instantiate our ideas and hypotheses in executable computational models, which can then be used to run valuable simulations and analyses—for example, Horschler et al. (2023) recently showed how a suite of executable models could be used to characterize the theory-of-mind of non-human primates. Finally, a systematic computational grounding for theories-of-mind can be applied to engineer real-world AI systems with human-like social intelligence in a principled manner.

The goal of this paper, then, is to offer a way to formalize θ and $\mathcal{T}$. We propose that we should think of a theory-of-mind θ as a kind of *programming language* (PL), one in which a mind can set up and solve problems involving reasoning about other minds. The space of theories-of-mind $\mathcal{T}$ is thus defined by a space of *potential* programming languages for reasoning about minds.

This paper develops this idea in four stages. We begin by discussing two prior attempts at formalizing theories-of-mind, and motivating why a new account would be valuable. Next, we introduce our account with an example of a *particular* programming language that instantiates a particular theory of mind. We use this example to show how a programming language's design can reify concepts in a theory of mind. We then move from the particular to the *general*, discussing how our account helps us formalize the differences between multiple possible theories-of-mind in terms of variations on a programming language's syntax and semantics. Finally, we discuss how our account gives insight into the process of theory change, by allowing us to draw wisdom from the history of programming language design.

Three formal accounts of theories-of-mind

What do we want from a formalization of a theory of mind? In our view, a formal account of theories-of-mind should at minimum be able to explain two of their key features: (1) how theories-of-mind empower their bearers to efficiently represent and reason about situations involving other minds, and (2) how these theories' bearers grow, mend, or replace theories entirely in the face of new evidence.

The Bayesian network account An early account (Gopnik et al., 2004; Gopnik & Wellman, 2012) addressed these desiderata by formalizing theories-of-mind as *causal Bayesian networks* (Pearl, 2000). On this account, latent concepts like an agent's beliefs and desires can be represented as nodes in a Bayesian network, with directed edges pointing to that agent's actions (Figure 1). Tasks like prediction and planning are supported by efficient algorithms for probabilistic inference (Baker et al., 2008). Finally, theories *themselves* can change in the face of new evidence, by hierarchical inference over a space of possible Bayesian networks (Goodman et al., 2006). For example, a child might start with high confidence that minds are well-modeled by the network shown in Figure 1(a). Over time, she might come to prefer the less-parsimonious network in Figure 1(b), because she discovers that it better models minds when perceptual access is limited.

This account faces two challenges. First, writing a Bayesian network to describe every possible social situation is simply untenable. One agent's beliefs and desires at a moment in time may be easy to represent with a fixed Bayesian network, but human capacity for reasoning about minds is unbounded, generative, and infinite: we can reason just as well about an agent's beliefs about *another* agent's beliefs, chang-

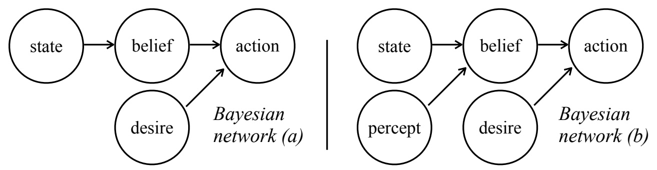

Figure 1: Two possible theories-of-mind, instantiated as Bayesian networks. Adapted from Goodman et al. (2006).

ing over time in response to that agent's actions. This suggests that a fixed Bayesian network is an insufficient representational scheme for theories of mind, challenging desideratum (1). Second, this account does not fully explain how one might *grow* a theory. Goodman et al. (2006)'s model considered inference between two predetermined Bayesian networks, but it did not account for where those hypotheses themselves may come from, challenging desideratum (2).

To begin addressing these challenges, Tenenbaum et al. (2007) proposed formalizing intuitive theories not as *fixed* Bayesian networks, but rather as *causal grammars* that generate a whole family of scenario-specific Bayesian networks. Causal grammars support reasoning about an infinitude of scenarios within a domain, and support learning via inference over hypotheses generated by the grammar. However, the formalism of a "grammar" is still too limited to represent the full richness of theory-of-mind—for example, to represent recursive beliefs about beliefs (Griffiths & Tenenbaum, 2007).

The programming pattern account A different approach was sparked by the development of *universal probabilistic programming languages* (PPLs) like Church (Goodman et al., 2012) and WebPPL (Goodman & Stuhlmüller, 2014). In a PPL, complex probabilistic models can be generated by simple programs: for example, repeated nodes in a Bayesian network can be generated automatically by loops.

Building on this idea, Goodman et al. (2015) cast theory-of-mind as a *toolbox of programming patterns* in a universal PPL. For example, on this account, representing an agent's rational goal-driven behavior amounts to the pattern of *planning-as-inference* (Botvinick & Toussaint, 2012), where to accomplish a goal an agent infers what actions "could have led" to the condition that the goal was achieved. Concretely, in WebPPL, we might write:

```
Infer({  // (a) planning as inference
  a = uniformDraw(Actions);
  condition(goal == step(state, a));
  return a;
})
```

Like causal grammars, these programming patterns enable programmers to flexibly generate situation-specific probabilistic models. However, these patterns are much more powerful than causal grammars. For example, representing "thinking about thinking" or recursive social reasoning amounts to the pattern of *nested inference:* running inference queries on probabilistic models which themselves recursively run inference queries (Stuhlmüller & Goodman, 2014; Zhang & Amin, 2022). Concretely, to infer an agent's `goal` given an `observed_action`, i.e. to perform Inverse Planning (Baker, Saxe, & Tenenbaum, 2009), we might write:

```
Infer({  // (b) inverse planning
  goal = uniformDraw(Goals);
  condition(observed_action == sample(
    Infer({  // nested Infer -- same as (a)
      a = uniformDraw(Actions);
      condition(goal == step(state, a));
      return a;
    })
  ));
  return goal;
});
```

This toolbox has in practice been remarkably successful, and has been used to quantitatively model a wide range of sophisticated phenomena in social cognition (Evans et al., 2017).

Have we finally rescued (1), then? Certainly, we have made progress on recovering the unbounded, generative, and infinite nature of human thought. But programming patterns are still not a satisfying formalization of a theory-of-mind, for two reasons: First, as we will show later in this paper, it is easy to write models that apply these programming patterns but whose behavior violates our intuitions—hence, the patterns must not fully capture key aspects of a typical theory-of-mind. Second, and more importantly, we have now lost (2), the capacity for theory change: there is no formal, enumerable schema of "programming patterns" that one might perform inference over when theorizing about minds.

The programming *language* account In this paper, then, we offer a new perspective that reconciles the strengths of the Bayesian-network/causal-grammar accounts and the programming-pattern account. On our account, a theory of mind is not like a set of programming patterns, but rather an entire programming *language*. Like Church and WebPPL, it is a probabilistic programming language. But unlike Church or WebPPL, it is not a *universal* PPL—rather, it is a *domain-specific* PPL (a "DSL"), specialized for the domain of reasoning about minds.

In the discipline of programming language theory, there is a long tradition of designing DSLs that capture the theories of various domains (Bentley, 1986; Bernstein & Kjolstad, 2016; Iverson, 2007), including scientific domains as in the theories of classical mechanics (Sussman & Wisdom, 2015), ray optics (Hanrahan & Lawson, 1990), and color perception (Chen, Chang, & Zhu, 2024). Instead of relying on the programmer to employ certain programming patterns, the syntax of a DSL *directly* provides the conceptual vocabulary for setting up problems in the target domain, and the semantics of such a language determine how problems are solved. The "theory" is thus represented by the design of the language itself.

Importantly, the syntax and semantics of any programming language can be mathematically formalized and even mechanized as members of a schema of many possible syntaxes and semantices (Felleisen, Findler, & Flatt, 2009; Pierce, 2002). Theory change can thus be modeled formally as learning over a *space of possible programming languages*.

What kinds of DSLs are we talking about, here?

Our discussion of DSLs has thus far has been very abstract. Before we continue, let us briefly introduce a concrete example of a DSL specialized for reasoning about reasoning: a real-world language called "memo" (Chandra, Chen, Tenenbaum, & Ragan-Kelley, 2025), whose name stands for "mental models." memo was developed as a practical tool for building efficient computational models of social cognition. In our discussion here, we will use memo (and, later, hypothetical variants of memo) as a way to illustrate examples, much like how Goodman et al. (2015) use the real-world language Church as a way to illustrate examples when discussing the Probabilistic Language of Thought hypothesis.

Let us start with a "crash course" on memo. Models in memo consist fundamentally of *agents* making *choices*. For example, suppose Ali chooses a bar b to visit after work one day. Because he is a rational agent, he is likelier to choose a nearby bar. In memo, we can model this choice using the **chooses** expression, saying that Ali chooses a bar b from the set of `Bars`, with probability proportional to ("wpp") some decreasing function of the distance from work to b:

```
# Example 1
Ali: chooses(b in Bars, wpp=1/dist(work, b))
```

memo then allows us to make various probabilistic queries about Ali's choice, such as the probability that one can find him at Tony's Tavern:

```
return Pr[ Ali.b == "Tony's␣Tavern" ]
```

Or the expected distance he will walk to get to the bar:

```
return E[ dist(work, Ali.b) ]
```

Now, suppose Zoe, who is at home, also goes to a nearby bar that evening. We can model these two agents separately and query the probability that they meet:

```
# Example 2
Ali: chooses(b in Bars, wpp=1/dist(work, b))
Zoe: chooses(b in Bars, wpp=1/dist(home, b))
return Pr[Ali.b == Zoe.b]  # chose same bar?
```

Suppose Zoe wishes to avoid Ali, choosing bar b with probability proportional to *her confidence* that Ali will not be at b (`Pr[b != Ali.b]`). To represent Zoe's beliefs about Ali's whereabouts, we can use memo's compound **thinks** statement, which sets up our model of *Zoe's* mental model of Ali:

```
# Example 3
Zoe: thinks[  # Zoe's mental model of Ali
  Ali: chooses(b in Bars, wpp=1/dist(work,b))
]
Zoe: chooses(b in Bars, wpp=Pr[b != Ali.b])
```

Of course, **thinks** can be nested arbitrarily: we can write "Zoe thinks Ali thinks Zoe thinks..." and so on.

Having set up these mental models, we can query the likelihood that Zoe will be at Tony's Tavern:

```
return Pr[ Zoe.b == "Tony's␣Tavern" ]
```

Or we can query our expectation of how worried Zoe will be (`Zoe[...]`) that she will meet Ali after all:

```
return E[ Zoe[ Pr[b == Ali.b] ] ]
```

(The outer expectation here is taken over the randomness of Zoe's choice of bar.)

As a last example, suppose that Zoe doesn't even know where Ali works—perhaps she only has uniform priors over a set of possible `Jobs`. We can amend her mental model of Ali to account for this additional hierarchical uncertainty:

```
Zoe: thinks[  # Example 4
  Ali: chooses(j in Jobs, wpp=1),  # uniform
  Ali: chooses(b in Bars, wpp=1/dist(j, b))
]
```

Zoe can still try to avoid Ali by marginalizing over her uncertainty over Ali's j. If she does nonetheless meet Ali, she can apply Bayes' rule to *infer* Ali's job ("you probably work near this bar"). To model this inference, we use the **observes** statement:

```
Zoe: chooses(b in Bars, wpp=Pr[b != Ali.b])
Zoe: observes Ali.b is Zoe.b  # they meet
```

Now, we can predict what Zoe's posterior belief will be, after meeting Ali, that he works at City Hall:

```
return E[ Zoe[ Pr[Ali.j == "City␣Hall"] ] ]
```

Why use a DSL at all? How can less be more?

memo statements like **chooses**, **observes**, and **thinks** correspond roughly to *sampling*, *conditioning*, and *recursive querying* in a universal probabilistic programming language. So what have we gained from writing these models in memo? In fact, haven't we given up expressive power by resorting to a specialized language—one that may in fact be ill-suited to express other kinds of programs?

An important lesson from decades of research in the field of programming languages is that universality comes at a cost: the burden of power. Making a language more expressive runs the risk of making it possible for programmers to express ideas they did not intend to express, which in turn increases the burden on the programmer to ensure that their programs are correct. Programming language pioneer Hudak (1996) thus wrote that the ideal abstraction for problem-solving is a programming language that exactly matches the domain of study: "no more, no less."

Church and WebPPL themselves are examples of this principle at work. In theory, by the logic of Turing-completeness, it is possible to implement by hand any WebPPL program's probabilistic calculation in any ordinary (non-probabilistic) programming language—in fact, WebPPL programs are executed by automatically translating them to programs in the programming language JavaScript (Goodman & Stuhlmüller, 2014). The value WebPPL adds over JavaScript is the guarantee that the probabilistic calculations will always be correct—for example, by guaranteeing that distributions will always be properly normalized. WebPPL empowers users to build complex probabilistic models by *limiting* them to only express correct reasoning about probability. Analogously, the value memo adds over WebPPL lies in its constraints: memo empowers users to build complex theory-of-mind models by limiting them to only express correct reasoning about agents.

Let us illustrate this with an example. Consider how we might implement the Ali-Zoe scenarios in WebPPL. Ali's choice of a nearby bar (Example 1) is straightforward to model by applying the planning-as-inference pattern (instead of **condition**, we use its counterpart **factor** to model a graded probabilistic choice):

```
ali_b_distribution = Infer({  // Example 1
  ali_b = uniformDraw(Bars);
  factor(log( 1/dist(work, ali_b) ));
  return ali_b;
})
```

So far, so good. What about Zoe's choice of a bar with the goal of avoiding Ali (Example 2)? Following the planning-as-inference pattern, it is tempting to write the following model:

```
zoe_b_distribution = Infer({  // Example 2(?)
  zoe_b = uniformDraw(Bars);
  ali_b = sample(ali_b_distribution);
  condition(zoe_b != ali_b);
  return zoe_b;
});
```

But, unexpectedly, this model produces the wrong answer. We can see the error most clearly if, instead of predicting Zoe's choice of bar, we predict the probability that they meet:

```
  return zoe_b == ali_b;
```

When we run this, we find that the model erroneously predicts a 0% chance that they meet. Ah, of course it does! Because we *conditioned* on `zoe_b != ali_b`, it is no surprise that the inference never meets that condition.

What has gone wrong here? The problem is that WebPPL has no language-level support for the notion of an *agent*—only the notion of a random choice. Zoe's choice of `zoe_b` and Ali's choice of `ali_b` are conceptually indistinguishable. Hence, when we apply the planning-as-inference pattern, WebPPL produces an inference as if Zoe could choose not only which bar she herself visits, but also which bar *Ali* visits—that is, as if Zoe's "wishful thinking" gives her "mind control" over Ali. Chandra, Chen, et al. (2025) refer to this type of bug as "perpetration confusion": confusion over which agents have agency over which random choices. It is quite common for even experienced programmers to encounter this bug in practice (Levine, 2018).

What this example illustrates is that while programming patterns in a universal PPL give us the building-blocks to express a vast range of computations that accord with our theory-of-mind, those same building-blocks can also be assembled to express computations that are *outside* our theory of mind—for example, computations that represent one agent having "mind control" over another. DSLs like memo address this by providing all (and only!) the constructs necessary for reasoning about minds: for example, by having language-level syntax and semantics for reasoning about agents, which by their very construction prevent issues of "mind control."

In the next section, we will study how exactly the design of a DSL like memo can instantiate a theory of mind.

$$
\begin{array}{rcl}
\langle \textit{statement} \rangle & ::= & a\colon\ \text{chooses}(x \text{ in } D, \text{wpp=}\langle \textit{expr} \rangle) \\
& | & a\colon\ \text{knows } b.x \\
& | & a\colon\ \text{observes } b.x \text{ is } c.y \\
& | & a\colon\ \text{thinks } [\langle \textit{statement} \rangle \ldots]
\end{array}
$$

Figure 2: Syntax of statements in memo. Variables a, b, c range over names of agents, and x, y over names of choices. Notice that the **thinks** statement has a uniquely recursive syntactic structure that itself references the ⟨*statement*⟩ rule. By omitting this rule from the syntax, we obtain a language that cannot represent false beliefs.

$$
\frac{e \text{ is well-typed} \qquad a \text{ has not already chosen } x \qquad \text{(...other premises omitted...)} \qquad \boxed{a \text{ knows } e}}{\texttt{a: chooses(x in X, wpp=e)} \text{ is well-typed}}\ \textsc{T-Chooses}
$$

Figure 3: Semantics of memo's **chooses** expression, simplified from Chandra, Amin, and Zhang (2025). In this notation, the premises above the bar allow deducing the conclusion below the bar. By omitting the boxed premise, we obtain a language whose semantics allow for "mind-reading," which can be used to formalize theories of omniscient minds.

How does a DSL instantiate a theory of mind?

A programming language is ultimately defined in terms of two formal structures: its *syntax* and its *semantics*. In this section, we will discuss through two case studies how memo's syntax and semantics reify principles in a theory of mind.

Along the way, we will show how *varying* memo's syntax and semantics gives rise to alternative theories of mind we may wish to formalize.[1] The study of theory of mind seeks to characterize not only the theory held by neurotypical adult humans, but also the theories-of-mind held by children, aging minds, and non-human primates. Furthermore, it seeks to characterize these minds' theories not only of ordinary minds, but also of "extraordinary" minds, such as omniscient beings and machines (Brink & Wellman, 2020), and it seeks to understand how minds decide *which* theory to attribute to an agent they observe in the world (Burger & Jara-Ettinger, 2020; Gray, Young, & Waytz, 2012). Variants of a DSL like memo can serve as formal representations of these theories.

A case study in syntax: false beliefs

In memo, the **thinks** statement is used to represent agents' beliefs. We have already seen how to use **thinks** to represent agents' true beliefs and uncertainty. But this construct can also be used to represent an agent's false belief. For example, to model that Zoe *thinks* Ali chooses a cheap bar, when he

[1]The notion of "language variants" is well-studied in PL. For example, some computer science curricula introduce programming through a progression of increasingly-powerful "student languages" (Felleisen, Findler, Flatt, & Krishnamurthi, 2018; Tsur & Rusk, 2018). Variants can also be used as executable models of students' *misconceptions* about a language (Chandra et al., 2024; Lu & Krishnamurthi, 2024).

actually chooses a nearby one, we might write:

```
Ali: chooses(b in Bar, wpp=1/dist(work, b))
Zoe: thinks[
  Ali: chooses(b in Bar, wpp=1/price(b))
]
```

Of course, the full generality of this statement is typically unnecessary. Most of the time, memo programmers simply need to express that an agent correctly knows another agent's choice. Suppose Zoe knows ahead of time where Ali is, perhaps because he had posted an update on social media. In principle, we could model this by writing that Zoe **thinks** Ali chooses a bar (from any distribution), and then correctly **observes** where he really went:

```
Ali: chooses(b in Bars, wpp=1/dist(work, b))
Zoe: thinks[
  Ali: chooses(b in Bars, wpp=...)
]
Zoe: observes Ali.b is Ali.b # observes truth
```

However, this is dissatisfying: it is verbose, and requires choosing a "dummy" prior distribution inside **thinks** which is never actually used for computation.

Because such situations are so common in everyday modeling, memo provides an efficient, lightweight shorthand to address these issues: the **knows** statement.

```
Ali: chooses(b in Bars, wpp=1/dist(work, b))
Zoe: knows(Ali.b)
```

memo's **knows** statement accords with Phillips et al. (2021)'s notion of "knowledge before belief": in many ways, **knows** is more fundamental than **thinks**/**observes**. Its syntax is simpler than the compound, recursive structure of **thinks** (see Figure 2), it is empirically used more frequently than either **thinks** or **observes**,[2] and it requires fewer lines of code to implement in the memo compiler.

Suppose then that we created a new "lite" variant of memo that dispenses with the complex **thinks** and **observes** statements, but retains the simpler **knows** statement. What would we lose? The resulting programming language, which we might call "memo-junior," could still be used to model a wide variety of scenarios, such as Examples 1 and 2 in the crash course above. However, "memo-junior" would not be able to represent scenarios where agents have their own beliefs that differ from the true state of the world. In this way, we could use memo-junior to begin to formalize a candidate hypothesis for the three-year-old child's theory of mind.

A case study in semantics: omniscience

Consider the following attempt at implementing Example 3:

```
Ali: chooses(b in Bars, wpp=1/dist(work, b))
Zoe: chooses(b in Bars, wpp=(b != Ali.b))
```

This model raises an error in memo—the problem is that Zoe is not aware of Ali's b in her mental model. It does not make sense for Zoe to choose a bar based on some function of Ali's b—in fact, she should not even be able to compute such a function in the first place! If Zoe could choose a bar this way, it would imply some kind of "mind-reading" of Ali on Zoe's part. Hence, memo forbids such statements.

But what exactly do we mean, formally, by "forbids" here? The precise rule can be found in memo's formal semantics (Chandra, Amin, & Zhang, 2025) as a precondition to deducing that a **chooses** statement is well-typed (Figure 3): an agent a must *know* the value of expression e if e is used to define the probability distribution with which a makes a choice. In practice, this precondition appears in the memo compiler as a check that is performed every time a **chooses** statement is encountered. If the check fails, an error message prompts the programmer to do what we did in Example 3: explicitly state Zoe's mental model of Ali's choice of bar, using either **knows** or **thinks** as needed.

In this way, the preconditions and postconditions of memo's semantics capture principles of agency, like the impossibility of mind-reading. The creators of memo explicitly enumerated the principles they intended to capture in the semantics, and explained in detail how memo's semantics reify them (Chandra, Chen, et al., 2025, see §3.1). But these principles were chosen by memo's creators so that the language matched *their* intuitions—they can easily be changed to accommodate other intuitions. For example, by relaxing the precondition discussed above, we can begin to formalize a theory-of-mind that includes the possibility of omniscient agents—a feature that seems to appear in children's theories of their parents (Barrett, Richert, & Driesenga, 2001).

How do theories—or DSLs—change over time?

Gopnik and Meltzoff (1997) liken the dynamics of theory-change to scientific theory change (Kuhn, 1962; Lakatos, 1970). But how can we formally model this process? Thinking of theories-of-mind as DSLs suggests casting theory construction as programming language *design*. In this section, we will discuss how this metaphor supports three perspectives on theory change.

The rational perspective Returning to Goodman et al. (2006)'s account of theory change as hierarchical inference over Bayesian networks, we might now say that theory change is driven by hierarchical inference over *syntaxes* and *semantices* of possible programming languages. We grow our theories in the direction of usefulness—representational efficiency and predictive power. This is, after all, how real-world programming languages tend to grow, too: not from "ivory-tower" innovations, but from demonstrated utility for solving actual problems (Meijer, 2007; Meyerovich & Rabkin, 2012; Stroustrup, 2020). How can we formalize this idea? Ellis et al. (2021) describe a system that grows DSLs by recognizing useful patterns and formalizing them into reusable abstractions. If we think of theories-of-mind as DSLs, we can imagine applying such a system to formalize the growth of theories-of-mind in the same way.

[2] Among the 24 memo models published in memo's documentation at the time of writing, **knows** was used 87 times while **thinks** and **observes** were used 43 and 39 times, respectively.

The cultural perspective Many lines of evidence suggest that learning natural language for theory-of-mind helps children acquire the ability to represent false beliefs (De Villiers & de Villiers, 2014; De Villiers & Pyers, 2002; Morgan & Kegl, 2006; Pyers, 2020; Pyers & Senghas, 2009; Schick, De Villiers, De Villiers, & Hoffmeister, 2007). For example, Pyers and Senghas (2009) write that "The sentence 'Mary thought she saw a ghost' has a main clause that is true ('Mary thought') and an embedded clause that is false ('she saw a ghost'). Such syntax may give learners the logical tools for understanding the false beliefs of others." The DSL perspective suggests a way to formalize this hypothesis. As we discussed earlier, we might model the 3-year-old child's theory-of-mind as a "memo-junior" that lacks the compound `thinks` expression. Exposure to the word "thinks" in the context of others' beliefs could supply the child with the recursive syntactic structure unique to `thinks`, which would enable the child to ultimately invent the *semantics* of `thinks` and thus to be able to represent false beliefs.

The pragmatic perspective Theory change is messy business. Gopnik and Meltzoff (1997) describe the liminal period when a new theory is gingerly considered, but the old theory is not yet abandoned. Indeed, Amsterlaw and Wellman (2006) find that a child on a given day might show evidence of using two or three different theories-of-mind across tasks.

Interestingly, PL design follows the same pattern: growing programming languages is messy business, too (Steele Jr, 1998). As an example, consider the case of the Python programming language. In the early 2000s, it became increasingly clear to Python users that the abstractions for working with text ("strings") were fundamentally flawed—the language failed to make the critical distinction between a sequence of characters and a sequence of bytes representing those characters. This made it difficult to process text input written in non-Latin alphabets, a serious problem for an increasingly global population of programmers. The authors of the language thus redesigned these abstractions from scratch in the next major version, Python 3, which was released in 2008 (van Rossum, 2008). We can think of this new version of the language as a kind of "paradigm shift."

But—just like with scientific paradigm shifts—the transition was not immediate (Coghlan, 2012). The old version, Python 2, remained actively used for the next *twelve years*, and was only formally phased-out in 2020. There were many reasons programmers hesitated to switch to Python 3 in the interim: some did not want to rewrite their existing Python 2 code in Python 3, while others depended on libraries only available in Python 2. Hence, for many years the choice between Python 2 and Python 3 was a pragmatic one: programmers embarking on new projects, or who really needed Python 3's features, generally chose the new version, while programmers working with large existing codebases continued to use Python 2 as long as they could. Many programmers even wrote their code carefully to be compatible with both versions at once (Malloy & Power, 2019).

We might expect a similar dynamic to govern the adoption of new theories-of-mind by children. Even though a new theory might be compelling, there might be pragmatic value in keeping around an older, tried-and-tested theory—perhaps to recycle models and computations that are only compatible with the old theory. A child might thus show patterns of behavior that, to an observing scientist, seem inconsistent with any one theory-of-mind.

Discussion and future work

In this paper, we proposed that it is valuable to think of theories-of-mind as *domain-specific programming languages* that one can use to set up and reason about problems involving other minds. Using the memo PPL as a case study, we discussed how a programming language can encode a theory-of-mind, how variants of a programming language can encode alternate theories-of-mind, and how the metaphor of a programming language can help us understand the development of theory-of-mind. We conclude by reflecting on a few more questions that PL theory may offer us tools to address:

Composing intuitive theories To solve real-world problems, our theories-of-mind must be able to interface with our intuitive theories of *other* domains, such as (for example) our intuitive theories of physics (Liu, Outa, & Akbiyik, 2024). How might we compose our intuitive theories to perform computations that straddle multiple different domain-specific systems of knowledge? If we treat intuitive theories as DSLs, then we can begin to reason about this question in terms of composing DSLs. In fact, the PL community has long studied how to design DSLs such that they can interface modularly with each other (see, e.g., Felleisen, Findler, Flatt, Krishnamurthi, Barzilay, et al., 2018). DSLs with well-designed "application programming interfaces" (APIs) allow programmers to write multilingual programs that combine the strengths of multiple DSLs. For example, to make a website, a programmer might use the language HTML to specify the page layout, the language CSS to specify the typography, and the language JavaScript to specify an animation. HTML, CSS, and JavaScript are very different languages, specialized to different domains of web design (structure, styling, and scripting, respectively). However, because all three DSLs come equipped with carefully-designed APIs, programmers can write programs that easily interface between them. In this way, we might imagine that our DSLs for reasoning about minds and our DSLs for reasoning about physics are equipped with suitable APIs, which allow them to be composed to reason about everyday socio-physical situations (see worked example here).

Comparing intuitive theories Intuitively, a human adult's theory-of-mind seems to be "richer" than that of a 3-year-old child or a non-human primate. But how can we formalize a claim about how "rich" or "expressive" a theory-of-mind is? Is it possible to quantify and measure the richness or expressivity of a theory? If we treat theories-of-mind as DSLs, then we can consider applying formal definitions of the "expressivity" of a programming language, such as the one proposed by Felleisen (1991), to formalize the expressivity of the theory-of-mind represented by that DSL. By using this definition as a measure of the expressivity of a given theory-of-mind, we could test the claim that the evolutionary and developmental trajectories of theories-of-mind tend towards greater expressivity over time.

Theor*ems* of mind People are not only capable of reasoning about specific situations involving other minds, they are also capable of reasoning about *general* statements about other minds. For example, we might agree that in *any* situation, if an agent forgets the value of a variable x, then they must previously have had a belief about x. Where do these general intuitions come from? If we treat theories-of-mind as DSLs, then we can formalize and study these "theorems of mind" by applying well-known techniques for stating and proving general theorems about a programming language. For example, Chandra, Amin, and Zhang (2025) show how to formally state the property that in *any* memo model, if an agent *knows* the value of a variable x, then they must be *certain* of x (i.e. from their perspective, $\mathcal{H}(x) = 0$). The converse does not hold, because an agent can be certain about a false belief. Hence, we can derive from first principles the "theorem of mind" that knowledge implies certainty, but certainty does not imply knowledge.

The frame problem Our full theory of other minds is vast, encompassing not only beliefs and desires, but also emotions, memory, attention, perception, and more. When faced with a concrete situation, how do we solve the "frame problem" (McCarthy & Hayes, 1981) of efficiently bringing to bear only the relevant part of our theory? Here, the PL notion of *program slicing* (Weiser, 1984) may be valuable: a *slice* of a program (or programming language) is a subset where irrelevant portions are removed. We might imagine that a rich, general theory-of-mind can be sliced into a bespoke situation-specific theory-of-mind that omits irrelevant syntax and semantics for the sake of efficiency.

Acknowledgements

We thank Rebecca Saxe and Elizabeth Spelke for many thoughtful conversations about these ideas. KC was supported by the Hertz Foundation and the NSF GRFP. JBT was supported by a Schmidt Sciences AI2050 Fellowship and the Siegel Family Quest for Intelligence.

References

Amsterlaw, J., & Wellman, H. M. (2006). Theories of mind in transition: A microgenetic study of the development of false belief understanding. *Journal of cognition and development*, *7*(2), 139–172.

Baker, C. L., Goodman, N. D., & Tenenbaum, J. B. (2008). Theory-based social goal inference. In *Proceedings of the thirtieth annual conference of the cognitive science society* (pp. 1447–1452).

Baker, C. L., Saxe, R., & Tenenbaum, J. B. (2009). Action understanding as inverse planning. *Cognition*, *113*(3), 329–349.

Barrett, J. L., Richert, R. A., & Driesenga, A. (2001). God's beliefs versus mother's: The development of nonhuman agent concepts. *Child Development*, *72*(1), 50–65.

Bentley, J. (1986). Programming pearls: little languages. *Communications of the ACM*, *29*(8), 711–721.

Bernstein, G. L., & Kjolstad, F. (2016). Perspectives: why new programming languages for simulation? *ACM Transactions on Graphics (TOG)*, *35*(2), 1–3.

Botvinick, M., & Toussaint, M. (2012). Planning as inference. *Trends in cognitive sciences*, *16*(10), 485–488.

Brink, K. A., & Wellman, H. M. (2020). Robot teachers for children? young children trust robots depending on their perceived accuracy and agency. *Developmental Psychology*, *56*(7), 1268.

Burger, L., & Jara-Ettinger, J. (2020). Mental inference: Mind perception as bayesian model selection. In *Cogsci.*

Carey, S. (1985). Conceptual change in childhood. *MIT Press.*

Chandra, K., Amin, N., & Zhang, Y. (2025). FOMO: a formal model of memo. *Languages for Inference (LAFI) workshop at POPL.*

Chandra, K., Chen, T., Tenenbaum, J. B., & Ragan-Kelley, J. (2025). A domain-specific probabilistic programming language for reasoning about reasoning (or: a memo on memo). *psyarxiv preprint*. Retrieved from `https://doi.org/10.31234/osf.io/pt863`

Chandra, K., Li, T.-M., Nigam, R., Tenenbaum, J., & Ragan-Kelley, J. (2024). Watchat: Explaining perplexing programs by debugging mental models. *arXiv preprint arXiv:2403.05334.*

Chen, E., Chang, J., & Zhu, Y. (2024). Coolerspace: A language for physically correct and computationally efficient color programming. *Proceedings of the ACM on Programming Languages*, *8*(OOPSLA2), 846–875.

Coghlan, A. (2012). Python 3 q-and-a.

De Villiers, J. G., & de Villiers, P. A. (2014). The role of language in theory of mind development. *Topics in Language Disorders*, *34*(4), 313–328.

De Villiers, J. G., & Pyers, J. E. (2002). Complements to cognition: A longitudinal study of the relationship between complex syntax and false-belief-understanding. *Cognitive development*, *17*(1), 1037–1060.

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

Evans, O., Stuhlmüller, A., Salvatier, J., & Filan, D. (2017). *Modeling Agents with Probabilistic Programs.* http://agentmodels.org. (Accessed: 2024-3-8)

Felleisen, M. (1991). On the expressive power of programming languages. *Science of computer programming*, *17*(1-3), 35–75.

Felleisen, M., Findler, R. B., & Flatt, M. (2009). *Semantics engineering with plt redex*. Mit Press.

Felleisen, M., Findler, R. B., Flatt, M., & Krishnamurthi, S. (2018). *How to design programs: an introduction to programming and computing*. MIT Press.

Felleisen, M., Findler, R. B., Flatt, M., Krishnamurthi, S., Barzilay, E., McCarthy, J., & Tobin-Hochstadt, S. (2018). A programmable programming language. *Communications of the ACM*, *61*(3), 62–71.

Goodman, N., Baker, C. L., Bonawitz, E. B., Mansinghka, V. K., Gopnik, A., Wellman, H., ... Tenenbaum, J. B. (2006). Intuitive theories of mind: A rational approach to false belief. In *Proceedings of the twenty-eighth annual conference of the cognitive science society* (Vol. 6).

Goodman, N., Gerstenberg, T., & Tenenbaum, J. B. (2015). Concepts in a probabilistic language of thought. In E. Margolis & S. Laurence (Eds.), *The conceptual mind: New directions in the study of concepts.* doi: https://doi.org/10.7551/mitpress/9383.003.0035

Goodman, N., Mansinghka, V., Roy, D. M., Bonawitz, K., & Tenenbaum, J. B. (2012). Church: a language for generative models. *arXiv preprint arXiv:1206.3255.*

Goodman, N., & Stuhlmüller, A. (2014). *The Design and Implementation of Probabilistic Programming Languages.* http://dippl.org. (Accessed: 2025-1-11)

Gopnik, A., Glymour, C., Sobel, D. M., Schulz, L. E., Kushnir, T., & Danks, D. (2004). A theory of causal learning in children: causal maps and bayes nets. *Psychological review*, *111*(1), 3.

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

Gopnik, A., & Wellman, H. M. (1992). Why the child's theory of mind really is a theory.

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

Gray, K., Young, L., & Waytz, A. (2012). Mind perception is the essence of morality. *Psychological inquiry*, *23*(2), 101–124.

Griffiths, T. L., & Tenenbaum, J. B. (2007). Two proposals for causal grammars. *Causal learning: Psychology, philosophy, and computation*, 323–345.

Hanrahan, P., & Lawson, J. (1990). A language for shading and lighting calculations. In *Proceedings of the 17th annual conference on computer graphics and interactive techniques* (pp. 289–298).

Horschler, D. J., Berke, M. D., Santos, L. R., & Jara-Ettinger, J. (2023). Differences between human and non-human primate theory of mind: Evidence from computational modeling. *bioRxiv*, 2023–08.

Hudak, P. (1996). Building domain-specific embedded languages. *Acm computing surveys (csur)*, *28*(4es), 196–es.

Iverson, K. E. (2007). Notation as a tool of thought. In *Acm turing award lectures* (p. 1979).

Kuhn, T. S. (1962). *The structure of scientific revolutions*. University of Chicago Press.

Lakatos, I. (1970). History of science and its rational reconstructions. In *Psa: Proceedings of the biennial meeting of the philosophy of science association* (Vol. 1970, pp. 91–136).

Levine, S. (2018). Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909.*

Liu, S., Outa, J., & Akbiyik, S. (2024). *Naive psychology depends on naive physics.* PsyArXiv. Retrieved from https://doi.org/10.31234/osf.io/u6xdz

Lu, K.-C., & Krishnamurthi, S. (2024). Identifying and correcting programming language behavior misconceptions. *Proceedings of the ACM on Programming Languages*, *8*(OOPSLA1), 334–361.

Malloy, B. A., & Power, J. F. (2019). An empirical analysis of the transition from python 2 to python 3. *Empirical Software Engineering*, *24*, 751–778.

McCarthy, J., & Hayes, P. J. (1981). Some philosophical problems from the standpoint of artificial intelligence. In *Readings in artificial intelligence* (pp. 431–450). Elsevier.

Meijer, E. (2007). Confessions of a used programming language salesman. *ACM SIGPLAN Notices*, *42*(10), 677–694.

Meyerovich, L. A., & Rabkin, A. S. (2012). Socio-plt: Principles for programming language adoption. In *Proceedings of the acm international symposium on new ideas, new paradigms, and reflections on programming and software* (pp. 39–54).

Morgan, G., & Kegl, J. (2006). Nicaraguan sign language and theory of mind: The issue of critical periods and abilities. *Journal of Child Psychology and Psychiatry*, *47*(8), 811–819.

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

Phillips, J., Buckwalter, W., Cushman, F., Friedman, O., Martin, A., Turri, J., ... Knobe, J. (2021). Knowledge before belief. *Behavioral and Brain Sciences*, *44*, e140.

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

Pyers, J. E. (2020). Constructing the social mind: Language and false-belief understanding. In *Roots of human sociality* (pp. 207–228). Routledge.

Pyers, J. E., & Senghas, A. (2009). Language promotes false-belief understanding: Evidence from learners of a new sign language. *Psychological science*, *20*(7), 805–812.

Schick, B., De Villiers, P., De Villiers, J., & Hoffmeister, R. (2007). Language and theory of mind: A study of deaf children. *Child development*, *78*(2), 376–396.

Steele Jr, G. L. (1998). Growing a language. In *Object-oriented programming, systems, languages, and applications (oopsla).*

Stroustrup, B. (2020). Thriving in a crowded and changing world: C++ 2006–2020. *Proceedings of the ACM on Programming Languages*, *4*(HOPL), 1–168.

Stuhlmüller, A., & Goodman, N. D. (2014). Reasoning about reasoning by nested conditioning: Modeling theory of mind with probabilistic programs. *Cognitive Systems Research*, *28*, 80–99.

Sussman, G. J., & Wisdom, J. (2015). *Structure and interpretation of classical mechanics*. The MIT Press.

Tenenbaum, J. B., Griffiths, T. L., & Niyogi, S. (2007). Intuitive theories as grammars for causal inference. *Causal learning: Psychology, philosophy, and computation*, 301–322.

Tsur, M., & Rusk, N. (2018). Scratch microworlds: designing project-based introductions to coding. In *Proceedings of the 49th acm technical symposium on computer science education* (pp. 894–899).

van Rossum, G. (2008). *What's new in python 3.0.* https://docs.python.org/3/whatsnew/3.0.html.

Weiser, M. (1984). Program slicing. *IEEE Transactions on software engineering*(4), 352–357.

Zhang, Y., & Amin, N. (2022). Reasoning about "reasoning about reasoning": semantics and contextual equivalence for probabilistic programs with nested queries and recursion. *Proceedings of the ACM on Programming Languages*, *6*(POPL), 1–28.