Assembly Instructions for the Modular Mind

Kartik Chandra[1], **Jonathan Ragan-Kelley**[1], **Joshua B. Tenenbaum**[2] **& Rebecca Saxe**[2]
[1]Computer Science and Artificial Intelligence Laboratory, MIT
[2]Department of Brain and Cognitive Sciences, MIT

Abstract

How do domain-specific cognitive systems interface with knowledge from beyond their respective domains? In this paper, we draw on the notion of domain-specificity in programming language theory to provide a computational account of how domain-specific systems might interface with domain-general world knowledge, as well as with other domain-specific systems.

Introduction

One of the most influential ideas in cognitive science is that some of our cognitive abilities are specialized to certain domains of thought (Carey, 2009; Fodor, 1983; Hirschfeld & Gelman, 1994; Spelke, 2022). For example, researchers have long argued that we have specialized systems for reasoning about objects ("intuitive physics") and agents ("intuitive psychology") (Wellman & Gelman, 1992), a view that has since been supported by many lines of converging evidence from cognitive development and cognitive neuroscience (see Liu, Karakose-Akbiyik, Outa, & Kim, 2025, for a recent review).

But nature rarely hands us problems that fit cleanly into one domain. Everyday life demands that we integrate our knowledge of objects and agents, and everything else we know, to solve the problems we are faced with. To decide whether to pass a football, we have to reason about the mind of our teammate, the dynamics of the ball, and the rules of game.

Our goal in this paper is to offer a computational account of how domain-specific systems of thought might interact with domain-general world knowledge, and with each other, to perform such reasoning. To give this account, we will draw from another discipline concerned with the notion of domain-specificity: the discipline of programming language theory (PLT). Most programming languages, like Python and C++, are general-purpose and can be used to write any program. But some programming languages are specialized for writing particular types of programs. These "domain-specific languages" (DSLs) are valuable because they are focused: they offer specialized syntax that helps programmers write programs that are succinct, correct, and efficient (Bentley, 1986).

Recently, we proposed that we can think of the domain-specialized system for intuitive psychology as a type of DSL: a programming language specialized for setting up and solving problems involving other minds (Chandra, Ragan-Kelley, & Tenenbaum, 2025). We concretely implemented this proposal using **memo**, a real-world DSL specialized for building theory-of-mind models via syntactic constructs like `knows`, `wants`, and `thinks` (Chandra, Chen, Tenenbaum, & Ragan-Kelley, 2025). Treating memo as a formal specification of intuitive psychology, we showed how this perspective can clarify a variety of theoretical issues in theory of mind research.

Here, we will extend this idea to study the domain-specificity of intuitive psychology. PLT has long studied how to design DSLs that can interface with external information, as well as with other DSLs. In the first part of this paper, we show how lessons from PLT clarify how a domain-specialized system for intuitive psychology might interface with domain-general world knowledge. Then, we will show how this perspective can help us understand how two domain-specialized systems might interface with one another.

The trouble with domain-specificity

A key theoretical challenge to domain-specificity, famously raised by Fodor (1983, p. 103), is that it is not clear that there can be modular, domain-specific cognitive systems at all. Fodor's concern is that the essence of modularity is the encapsulation of information: modules should have limited access to external knowledge. But on Fodor's view, information encapsulation is incompatible with the work of cognitive systems. Cognitive systems are responsible for belief fixation, and belief fixation by its very nature requires unfettered global access to information, because any bit of world knowledge could potentially be relevant to the problem at hand. "In principle, our botany constrains our astronomy," writes Fodor—or, specializing this argument to the domain of social cognition, Currie and Sterelny (2000) write: "Just where are the bits of information to which we are systematically blind in making our social judgements? The whole genre of the detective story depends on our interest and skill in bringing improbable bits of far-away information to bear on the question of someone's credibility." A domain-specialized system is not useful without a mechanism for applying that system to reason about arbitrary entities in the open world. But it is not clear how to design a system that encapsulates its domain-specific knowledge while operating over general world knowledge—or whether such a system, if built, would end up counting as domain-specific at all.

This issue is reflected in traditional computational models[1] of theory-of-mind reasoning. Consider for example

[1]Here, we are concerned specifically with the challenge of building computational models that reflect modular organization of *knowledge* in the mind. The possible modularity of perception, motor

In G. Boleda, I. Dautriche, M. Maldonado, C. Saldana and T. Torrent (Eds.), *Proceedings of the 48th Annual Conference of the Cognitive Science Society*

Baker, Jara-Ettinger, Saxe, and Tenenbaum (2017)'s well-known Bayesian inverse planning model of belief and desire attribution in the "food trucks" paradigm. This model takes as input a cartoon video of a hungry graduate student searching for a food truck, and produces judgments of the graduate student's preferences over food truck cuisine options, as well as the graduate student's initial (possibly-false) beliefs about where various food trucks are parked. The outputs of this model predict human judgments very well, suggesting that the model captures important aspects of how people reason about these scenarios.

But, taking a step back, what exactly is Baker et al.'s model a model *of?* In principle, it is a model of theory-of-mind reasoning, and indeed, the model's source code includes procedures for performing computations that should clearly be considered part of theory-of-mind under the Bayesian inverse planning framework (e.g. procedures for computing noisily-rational actions and for performing Bayesian belief updates). But the source code to this model is also peppered with procedures for a wide variety of *other* computations, computations that seem to have nothing to do with theory-of-mind per se: for example, procedures for computing the walking distance between two parking spots, or reasoning about the impenetrability of a brick wall. These food-truck-and-campus-layout-specific procedures are interleaved with the source code of the full Bayesian inverse planning model, tucked between procedures for computing noisily-rational action and Bayesian belief updates. This hardly seems like a model of a modular system: there is no clear boundary that demarcates where the theory of mind ends and the theory of parking spots begins. In this way, Baker et al.'s model excels as a model of how people reason specifically about graduate students foraging at food trucks, but it is not a model of theory-of-mind itself.

A Fodorean skeptic might say that this is no surprise: in fact, it is impossible to deliver such a model. Any effort to build a computational model of theory-of-mind itself is condemned to fail in one of two ways. One failure mode is for the model to be so "pure" as to be unable to account for any concrete real-world phenomena. After all, without building in some notion of distances and walls into the food trucks model, it would be impossible to reason about the relative costs of the graduate student's potential routes, and thus impossible to apply the naïve utility calculus to make mental state inferences. The other failure mode is including so much non-theory-of-mind-related world knowledge as to defeat any plausible claims of modularity.

Can we reconcile the empirical evidence in favor of a domain-specialized system with the theoretical challenge raised by Fodor? We propose to do so by formalizing theories-of-mind not as programs but as programming *languages*. In fact, "domain-specificity," "modularity," and "encapsulation" are all common terms in the programming languages literature (see, e.g., Bentley, 1986; Boyapati, Liskov, & Shrira, 2003; Hudak, 1998; Leroy, 1994; Parnas, 1972, for some examples of seminal papers on these topics). Modular software architectures have many pragmatic benefits: modules can be reused across projects, they can be checked for correctness in isolation, they can be developed independently and in parallel by different teams, and they can be used by non-specialists who are agnostic to the technical implementation of the module's internals. Every DSL designer faces the challenge of designing a system that cleanly abstracts away the particulars of its respective domain, while still being useful in the real world. Examining how DSL designers navigate this tension can help us understand how domain-specific cognitive systems may be architected to participate in domain-general computations.

control, and memory, which have long been reflected in cognitive architectures like ACT-R (Anderson, 2009) and Soar (Laird, Newell, & Rosenbloom, 1987), are complementary axes of modularity in the mind.

Domain-general knowledge in DSLs

To study how DSL designers navigate this tension, let us consider as a case study the Structured Query Language (SQL), a widely-used DSL that is specialized for making queries to databases. SQL's core competence is in helping programmers perform database search tasks like "search this database of real estate listings for the 10 cheapest 2-bedroom apartments within a mile of the nearest train station" or "search this database of student records for students enrolled in Calculus without the prerequisite of Algebra." In the same way that memo is a formalization of a theory of mind, which is used to efficiently represent agents' mental states and predict their behavior, SQL is a formalization of a theory of structured relational data (Codd, 1970), which is used to efficiently represent database schemas and execute queries over them. Just as memo has built-in algorithms for performing Bayesian inference about hidden mental states, SQL's compiler has built-in algorithms for efficiently sorting, indexing, and filtering data.

To answer a query about 2-bedroom apartments, a SQL program must be able to reason over concepts of apartments and bedrooms—at the very least, to understand that every apartment is associated with some number of bedrooms. This knowledge is clearly beyond the scope of the general theory of structured relational data, and a domain-specific language for database queries should not be endowed with bespoke knowledge of bedrooms (nor should a domain-specific system for reasoning about other agents be endowed with bespoke knowledge of food trucks). SQL must thus be designed so that without any such built-in knowledge, it is able to answer queries about apartments and bedrooms (not to mention employees and managers, books and authors, courses and instructors, and all other facets of the ever-changing real world). How is this possible? SQL accomplishes this by requiring programmers to provide the relevant world knowledge themselves. SQL programmers are responsible for defining the ontology of relevant data types and relations when configuring their database schemas, as well as for defining helper functions (e.g. for computing the walking distance between an apartment and the nearest train station) when performing queries. Equipped with this bespoke, curated slice of world

knowledge, SQL can then draw on its core competence in processing structured relational data to efficiently represent and execute the necessary database queries.

The lesson to learn, then, is that domain-specific systems can be designed to flexibly accept incoming exogenous world knowledge on an ad-hoc, situation-specific basis. This lesson echoes theoretical work in cognitive science on the issue of modularity and encapsulation. For example, responding to Currie and Sterelny's example of the detective story, Carruthers (2006) argues that "All that the example really shows is that the mind-reading faculty may need to work in conjunction with other elements of cognition in providing us with a solution to a problem, querying other systems for information. In fact," he continues, "most of the burden in detective-work is placed on physical enquiries of one sort or another—investigating foot-prints, finger-prints, and so forth." Carruthers advocates for distinguishing between "narrow-scope" and "wide-scope" notions of encapsulation. In both cases, the encapsulated system has native internal information specific to its domain. But they differ in the degree to which they can access external information. A narrowly encapsulated system can only access a small, *determinate* subset of the available external information, while a widely encapsulated system can be dynamically configured to query *any* small subset of the total available external information. In principle, wide-scope encapsulation retains the key computational benefit of narrow-scope encapsulation—namely, what Carruthers calls "computational frugality," in the sense of limited use of computational resources—but in addition, it makes it possible for cognitive systems to be modular after all.

Our proposal is that DSLs provide a concrete computational account of cognitive systems with wide-scope encapsulation, and thus of systems that are domain-specialized while being able to work with general world knowledge. Let us return to the domain of social cognition, as modeled by the DSL memo, for an example. How would we implement Baker et al.'s food trucks model in memo? The full model is available online[2], but let us focus on a couple of key highlights here. First, the model sets up some basic data types that are necessary to reason over this scenario, such as the space S of possible locations on a 2D grid, and the space A of possible actions that the hungry grad student can take at any given moment. To be clear, this is *not* memo code, but rather ordinary Python code that we will reference in memo shortly.

```
S = pair(x=range(0,WIDTH),y=range(0,HEIGHT))
class A(Enum):
  LEFT = 0; RIGHT = 1; UP = 2; DOWN = 3
```

A helper function then describes how taking an action $a \in A$ changes the grad student's location from $s \in S$ via vector addition. (The full code includes a little more logic to account for the impenetrability of walls.)

```
def take_a_step(s, a):
  deltas = [[-1,0], [+1,0], [0,-1], [0,+1]]
  return S(s + deltas[a])
```

Having defined this basic "general world knowledge," we can now use memo to model the grad student taking a step in the direction that maximizes the chances that (for example) she ends up in a parking spot where there is food.

```
student: knows(s)
student: wants(goal=food_at_state(s'))
student: chooses(a in A,
    to_maximize=EU[goal])
student: given(s' in S, wpp=s' ==
    take_a_step(s, a))
```

Notice that the memo code is agnostic to how S, A, and `take_a_step` are defined, or what they do. That "general world knowledge" was expressed in Python, a general-purpose programming language. memo is however able to accept this incoming exogenous information and integrate it into the model in order to perform theory-of-mind-specific reasoning over this world knowledge.

The Fodorean skeptic might still object that we have not really made any progress with this model. After all, isn't this model ultimately just like Baker et al.'s, in the sense that code modeling parking-lot-geometry is juxtaposed with code modeling rational action? And doesn't the memo code still need to know at least about the *existence* of S, A, and `take_a_step`—which means it is not truly a modular system after all? Here, it is important to remember that this code is *not* meant to be a model of theory-of-mind. This code is indeed a model of how people reason about hungry grad students foraging at food trucks. The theory-of-mind itself is modeled by code that is not even pictured here: namely, the code that implements memo's *compiler*. The compiler is where the implementations of noisily-rational action and Bayesian belief updating lie: it is shared unmodified among all memo users and all memo models, and it is clearly agnostic to even the existence of S, A, and `take_a_step`. The memo compiler knows how to work with agents, action spaces, and utility functions (concepts that are posited as part of the theory of mind), but it has no built-in knowledge of any *particular* agents, action spaces, or utility functions. Those must be provided by memo programmers. In this way, the memo compiler is a domain-specialized and informationally-encapsulated system that is nonetheless able to interface with general world knowledge.

Composing domain-specialized systems

In our discussion so far, we have used facts like "the impenetrability of walls" as examples of exogenous world knowledge that is outside of theory-of-mind. But the impenetrability of solid objects is not an isolated fact about the world—rather it itself is a key principle of a different intuitive theory: an intuitive theory of physics. Let us turn, then, to the question of how a domain-specialized cognitive system could interface, not just with isolated bits of exogenous knowledge, but with another domain-specialized cognitive system.

Consider the case of intuitive psychology and intuitive physics. As Liu et al. (2025) discuss, there is substantial

[2] https://osf.io/x4uak/overview?view_only=01eaf2089bbf4eb1982b2b764977f798

evidence that people have separate domain-specialized systems for reasoning about the mental states of agents in the social world, and for reasoning about the dynamics of objects in the physical world. But in most real-world situations involving other people, we must simultaneously reason about both of these types of entities in a deeply integrated manner. For example, in Csibra, Bíró, Koós, and Gergely (2003)'s classic experiments, one-year-old infants infer the presence of a barrier when seeing an agent jumping to reach a target object. To make this inference, the infant must reason both about the agent's physical body (e.g. the effortfulness of jumping, the impenetrability of solid barriers) and the agent's mental states (e.g. the principle of rational action (Jara-Ettinger, Gweon, Schulz, & Tenenbaum, 2016), by which the agent is unlikely to take an effortful and costly jumping action in the absence of a barrier). How do intuitive physics and intuitive psychology interface with one another to solve this problem?

To approach this issue from a DSL perspective, let us treat these two different domain-specialized cognitive systems (intuitive physics and intuitive psychology) as two different DSLs. Thus far, we have focused on treating intuitive psychology as a memo-like DSL, but implicit in our proposal has been the idea that *any* intuitive theory could be thought of as a DSL. Indeed, a growing body of work argues that rather than thinking about "the" language of thought, we should think about the many different domain-specific languag*es* of thought present in a given mind (Dehaene, Al Roumi, Lakretz, Planton, & Sablé-Meyer, 2022; Mandelbaum et al., 2022; Quilty-Dunn, Porot, & Mandelbaum, 2023; Carruthers, 2006, §6.3). The history of programming languages is full of examples of DSLs that formalize theories of all kinds of domains, ranging from the theories of scientific domains like classical mechanics (Sussman & Wisdom, 2015) to ray optics (Hanrahan & Lawson, 1990), color perception (Chen, Chang, & Zhu, 2024), and differential geometry (Li, Kamil, Crane, Jacobson, & Gingold, 2024), to the theories of culturally-constructed activities like knitting (Lin et al., 2023) and music (Anders & Miranda, 2011).

If we adopt this perspective, then the question becomes: Is it possible to write "multilingual" (Felleisen et al., 2018) programs that combine multiple domain-specific languages to solve domain-general problems? Programming language theory tells us that the answer is yes. For example, to build a website, a programmer might use the language PHP to maintain the web server, the language SQL to query the database, the language HTML to specify the page layout, the language CSS to design the fonts and colors, and the language JavaScript to implement the behaviors of interactive buttons. These five languages are specialized to their various domains, but they work together in concert. When a visitor to a website clicks on a button, a bit of JavaScript code is activated, which "talks to" a bit of PHP code on the web server, which "talks to" some SQL code in order to query the database, and finally the result is passed back from SQL to PHP to JavaScript and displayed to the visitor by "talking to" the HTML and CSS.

How is this multilingual interaction implemented? Programming languages talk to one another using two paired mechanisms. A "foreign function interface" (FFI) allows code in programming language *A* to make outgoing requests to run code written in programming language *B*, while an "application programming interface" (API) allows code written in programming language *B* to accept incoming requests to run code from programming language *A*. For example, to query a database from a web server, a programmer might connect PHP's FFI to SQL's API.

Let us see how FFIs and APIs can be used to build a model that reasons about an agent jumping to reach a target object, as in Csibra et al.'s stimuli, by combining memo with a different DSL that captures intuitive physics.

Precisely characterizing this putative language-of-thought-of-physics is beyond the scope of this paper, but let us briefly describe what such a language might look like. One way intuitive physics has been characterized in computational terms is by analogy to a "video game engine," a system that supports setting up and efficiently simulating "levels" of a game (i.e. physical scenarios) based on resource-rational approximations to the laws of physics (Battaglia, Hamrick, & Tenenbaum, 2013; T. D. Ullman, Spelke, Battaglia, & Tenenbaum, 2017). Perhaps unsurprisingly, video game designers have long developed DSLs that make it easier to set up games and levels. These languages have evolved with advancements in computer graphics, but almost always revolve around specifying objects and their causal relationships. Very early text-based games like *Zork* were implemented using languages like Zork Implementation Language (ZIL) (Meretzky, 1989) and Inform (Nelson, 1993), which provide syntax for declaring objects, their locations and spatial relationships (e.g. support or containment), and abstract rules for how their properties change in response to interactions with other objects. Primitive graphical games, including Atari-like games like *Frogger*, can be specified using languages like Video Game Description Language (VGDL) (Ebner et al., 2013; Schaul, 2013) or PuzzleScript (Lavelle, 2013), which provide syntax for specifying the spatial positions of objects on a 2D grid, as well as how these objects move and interact upon contact with one another. Finally, modern video games with rich, continuous-space rigid-body dynamics, like *Angry Birds*, are implemented using frameworks like box2d (Catto, 2007), which provide syntax for specifying objects and their positions, velocities, and physical properties, as well as tools for simulating Newtonian mechanics on these scenes. In the same way that the syntax and semantics of memo formalize psychological concepts like agents, beliefs, and desires, the syntax and semantics of languages like ZIL, Inform, VGDL, PuzzleScript, and box2d formalize physical concepts like bodies, positions, contact, containment, forces, and torques. In this way, we might take these languages to be formalizations of intuitive theories of physics.

To model people's intuitions about Csibra et al.'s stimuli, we will use a physics engine called JAX2d (Matthews, Beukman,

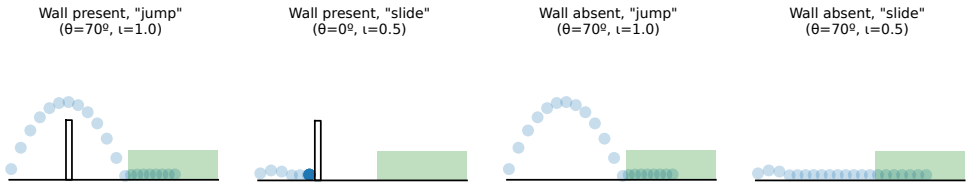

Figure 1: Sample simulations from our JAX2d simulator, varying the impulse ι and the angle θ to produce "jump" and "walk" motions, in the presence and absence of an immovable wall.

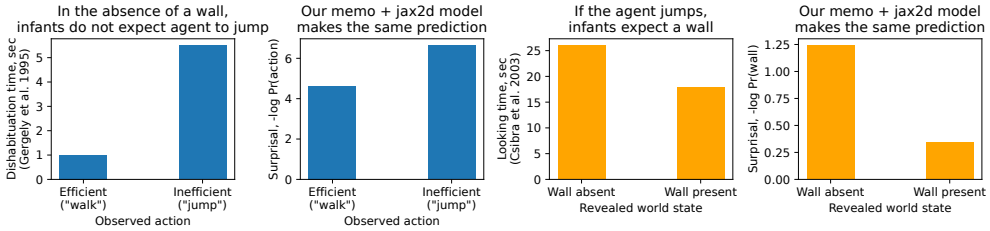

Figure 2: Like 12-month-old infants in two studies, our combined memo+jax2d model expects **(left)** agents not to jump in the absence of a wall (Gergely et al., 1995), and **(right)** a wall to be present if the agent jumped (Csibra et al., 2003). Here, infant intuitions are measured in dishabituation time and looking time (in seconds), while model predictions are measured in surprisal.

Lu, & Foerster, 2025), which is a modern, lightweight variant of box2d. JAX2d has an API that lets us manipulate JAX2d scenes using Python code. For example, in JAX2d, we can set up a scene with an agent who has a circular body like this:

```
scene = create_empty_sim()
add_circle_to_scene(
  scene, radius=0.1, friction=0, density=1,
  position=[0.0, 0.1], velocity=[0.0, 0.0],
)
```

Similarly, we can conditionally install an immovable wall by adding a rectangular body:

```
if wall:
  add_rectangle_to_scene(
    scene, fixated=True,  # immovable
    position=[1., .5], dimensions=[.1, 1.0],
  )
```

JAX2d's API exposes a Python function `step(scene)`, which steps the simulation of `scene` forward one tick in time. We can optionally supply external forces that act on each body in the scene. For example, we can write `step(sim_state, [0.0, 1.0])` to give a vertical "kick" to the agent, i.e. an impulse of 1.0 units in the $+Y$ direction. Putting this together with our scene setup code, we can write a simple function that simulates an agent moving by giving it an impulse ι in direction θ at time $t = 0$, and simulating for 100 more steps.

```
def simulate(wall, ι, θ):
  scene = create_empty_sim()
  agent = add_circle_to_scene(...)
  if wall: add_rectangle_to_scene(...)
  scene = step(scene, [ι·cos(θ), ι·sin(θ)])
  for t in range(100): scene = step(scene)
  return scene
```

Figure 1 shows some sample simulations generated by this function. Notice that by varying θ and ι, we can produce both "jumping" and "walking" actions.

Now, let us write a memo model that reasons about the agent by interfacing with this physics simulation. In our model, the `subject` in Csibra et al.'s experiment is unsure whether or not a wall is present. However, the subject thinks that the `agent` knows whether the wall is present. The subject models the `agent` as a rational actor who, by the principle of rational action (Jara-Ettinger et al., 2016), chooses an action (θ, ι) that maximizes the agent's utility. We model the agent's noisily-rational choice via the softmax choice rule (Franke & Degen, 2023; Luce, 1959).

```
subject: thinks[
  world: given(wall in Bool, wpp=1),
  agent: knows(world.wall)
  agent: chooses(
    θ in Angle, ι in Impulse,
    wpp=exp(β * utility(θ, ι, world.wall))
  )
]
```

What *is* the agent's utility? Utilities are given by costs and rewards. Here, let us say that the cost of an action is proportional to the magnitude of impulse needed, ι, and the reward is given by whether or not the agent is at the goal after the simulation. We can implement this utility as a Python function.

```
def utility(wall, ι, θ):
  scene = simulate(wall, ι, θ)
```

```
    reward = 1 if agent_at_goal(scene) else 0
    cost = ι
    return reward - cost
```

We can reference the function `utility` from our memo model because memo's FFI allows direct calls to Python functions. The function `utility` in turn calls `simulate` through JAX2d's API. This is how memo "talks to" JAX2d. Notice that it is not obvious that this should be possible at all: memo was not explicitly designed to be compatible with JAX2d, and JAX2d was not explicitly designed to be compatible with memo. In fact, these software packages were developed in isolation by teams that never communicated. Nonetheless, these individually-developed domain-specific systems are compatible because of their FFIs and APIs.

Let us now complete our model. The subject observes the agent's chosen action, and then makes an inference about the presence of the wall.

```
subject: observes [agent.θ, agent.ι]
return subject[Pr[world.wall]]
```

Figure 2 shows our model's predictions next to data from 12-month-old infants, showing the same qualitative predictions across two experiments. Consistent with Gergely et al. (1995)'s data, our model expects the agent not to jump in the absence of a wall, and consistent with Csibra et al. (2003)'s data, our model expects a wall if the agent jumps.

To be clear, there is nothing particularly novel about creating a model that makes such predictions. Many existing models can perform social reasoning grounded in physical interactions (see, e.g. Liu et al., 2017; Netanyahu et al., 2021; Shu et al., 2021; T. Ullman et al., 2009). But these models are typically thousands of lines of code long, densely interleaving implementations of physics simulations and forward- and inverse-planning algorithms in a general-purpose programming language. What is interesting about our model is its construction out of two modular components. Our JAX2d program modeled the agent's physical body, without regard to psychology, and our memo program modeled the agent's mind, without regard to physics. Importantly, we did not have to implement *theories* of physics or psychology—we never directly expressed Newton's laws or Bayes' rule—because these theories were already implemented in the respective compilers of JAX2d and memo.

Final assembly

In this paper, we offered an explanation of how a domain-specific system could in principle be supplied with exogenous world knowledge. But we have not yet addressed how the mind as a whole decides *which* exogenous world knowledge to supply to a domain-specific system. We sidestepped this problem by hand-curating the world knowledge that was supplied to the model: for example, in the food trucks model, we hand-wrote the definitions of S, A, and `take_a_step` up-front. But let us now consider where this world knowledge might come from. When faced with the food trucks problem, how does the mind determine that the model should include knowledge of the impermeability of brick walls, but not (for example) knowledge of brick chemistry or bricklaying patterns (the "frame problem" of McCarthy & Hayes, 1981)? And, for that matter, how does the mind decide to apply the domain-specific system for intuitive psychology in the first place?

One candidate answer comes from a recent line of work by Wong et al. (2025), who, like us, adopt the metaphor of thinking as writing and executing programs in a mental programming language. They propose that when the mind is faced with a new problem, it first draws on a global associative store to surface the relevant world knowledge, and then applies that world knowledge to program a situation-specific model to reason over. Wong et al. offer a concrete instantiation of this "model synthesis architecture" (MSA) by using a pretrained large language model (LLM)—a natural choice, because LLMs can function both as a global store of world knowledge (because they are trained on large swathes of the Internet), and as proficient program synthesis systems (because they are trained specifically on code). When faced with a particular problem, the MSA uses the LLM's language-to-code facilities to program a situation-specific model, which can be executed to generate predictions and inferences. The MSA generates human-like predictions about a variety of different "open-world" scenarios. For example, by drawing on information about sports encoded in the LLM's weights, the MSA can reason about the outcomes of arbitrary sporting events on demand, without requiring model designers to curate bespoke knowledge of those sporting events ahead of time.

Can an MSA be extended to recruit, not just isolated bits of world knowledge, but an entire intuitive theory, to solve a problem? On our account, an MSA can choose to apply an intuitive theory by choosing to program its model in the theory's respective DSL. For example, when faced with the food trucks problem, the MSA may determine by association that intuitive psychology is a relevant body of knowledge to bring to bear in solving the problem. It would then choose to deploy theory-of-mind by programming a model in memo, and it would supply the model with the requisite exogenous world knowledge in the manner described above.

Conclusion

Let us summarize our answer to Fodor's puzzle. How is it possible to have a system for social cognition that, on one hand, is domain-specialized, and, on the other hand, solves real-world problems? DSLs offer a way to formalize a theory of a domain in a way that is informationally encapsulated, but nonetheless allows programmers to supply incoming exogenous world knowledge. FFIs and APIs allow separate DSLs to interface with each other, explaining how the mind might combine intuitive theories. Finally, MSAs are a candidate model of how the mind might surface and apply the relevant world knowledge and intuitive theories in the first place.

Acknowledgments

We thank Elizabeth Spelke, Shari Liu, and the anonymous reviewers for thoughtful comments on these ideas. This work was supported by the MIT Siegel Family Quest for Intelligence, Schmidt Sciences, the Simons Foundation, and the Hertz Foundation.

References

Anders, T., & Miranda, E. R. (2011). Constraint programming systems for modeling music theories and composition. *ACM Computing Surveys (CSUR)*, *43*(4), 1–38.

Anderson, J. R. (2009). *How can the human mind occur in the physical universe?* Oxford University Press.

Baker, C. L., Jara-Ettinger, J., Saxe, R., & Tenenbaum, J. B. (2017). Rational quantitative attribution of beliefs, desires and percepts in human mentalizing. *Nature Human Behaviour*, *1*(4), 0064.

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.

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

Boyapati, C., Liskov, B., & Shrira, L. (2003). Ownership types for object encapsulation. *ACM SIGPLAN Notices*, *38*(1), 213–223.

Carey, S. (2009). *The origin of concepts*. Oxford University Press. Retrieved from https://doi.org/10.1093/acprof:oso/9780195367638.001.0001 doi: 10.1093/acprof:oso/9780195367638.001.0001

Carruthers, P. (2006). *The architecture of the mind*. Oxford University Press.

Catto, E. (2007). *box2d*. Retrieved from https://github.com/erincatto/box2d

Chandra, K., Chen, T., Tenenbaum, J. B., & Ragan-Kelley, J. (2025, October). A domain-specific probabilistic programming language for reasoning about reasoning (or: A memo on memo). *Proc. ACM Program. Lang.*, *9*(OOPSLA2). Retrieved from https://doi.org/10.1145/3763078 doi: 10.1145/3763078

Chandra, K., Ragan-Kelley, J., & Tenenbaum, J. (2025). Theories of mind as languages of thought for thought about thought.

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.

Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, *13*(6), 377–387.

Csibra, G., Bíró, S., Koós, O., & Gergely, G. (2003). One-year-old infants use teleological representations of actions productively. *Cognitive science*, *27*(1), 111–133.

Currie, G., & Sterelny, K. (2000). How to think about the modularity of mind-reading. *The Philosophical Quarterly*, *50*(199), 145–160.

Dehaene, S., Al Roumi, F., Lakretz, Y., Planton, S., & Sablé-Meyer, M. (2022). Symbols and mental programs: a hypothesis about human singularity. *Trends in Cognitive Sciences*, *26*(9), 751–766.

Ebner, M., Levine, J., Lucas, S. M., Schaul, T., Thompson, T., & Togelius, J. (2013). Towards a video game description language.

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.

Fodor, J. A. (1983). *The modularity of mind*. MIT press.

Franke, M., & Degen, J. (2023). The softmax function: Properties, motivation, and interpretation.

Gergely, G., Nádasdy, Z., Csibra, G., & Bíró, S. (1995). Taking the intentional stance at 12 months of age. *Cognition*, *56*(2), 165–193.

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).

Hirschfeld, L. A., & Gelman, S. A. (1994). *Mapping the mind: Domain specificity in cognition and culture*. Cambridge University Press.

Hudak, P. (1998). Modular domain specific languages and tools. In *Proceedings. fifth international conference on software reuse (cat. no. 98tb100203)* (pp. 134–142).

Jara-Ettinger, J., Gweon, H., Schulz, L. E., & Tenenbaum, J. B. (2016). The naive utility calculus: Computational principles underlying commonsense psychology. *Trends in Cognitive Sciences*, *20*(8), 589–604.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial intelligence*, *33*(1), 1–64.

Lavelle, S. (2013). *Puzzlescript*. Retrieved from https://github.com/increpare/puzzlescript

Leroy, X. (1994). Manifest types, modules, and separate compilation. In *Proceedings of the 21st acm sigplan-sigact symposium on principles of programming languages* (pp. 109–122).

Li, Y., Kamil, S., Crane, K., Jacobson, A., & Gingold, Y. (2024). I mesh: A dsl for mesh processing. *ACM Transactions on Graphics*, *43*(5), 1–17.

Lin, J., Narayanan, V., Ikarashi, Y., Ragan-Kelley, J., Bernstein, G., & McCann, J. (2023). Semantics and scheduling for machine knitting compilers. *ACM Transactions on Graphics (TOG)*, *42*(4), 1–26.

Liu, S., Karakose-Akbiyik, S., Outa, J., & Kim, M. J. (2025). How physical information is used to make sense of the psychological world. *Nature Reviews Psychology*, 1–15.

Liu, S., Ullman, T. D., Tenenbaum, J. B., & Spelke, E. S.

(2017). Ten-month-old infants infer the value of goals from the costs of actions. *Science*, *358*(6366), 1038–1041.

Luce, R. D. (1959). *Individual choice behavior* (Vol. 4). Wiley New York.

Mandelbaum, E., Dunham, Y., Feiman, R., Firestone, C., Green, E., Harris, D., ... others (2022). Problems and mysteries of the many languages of thought. *Cognitive Science*, *46*(12), e13225.

Matthews, M., Beukman, M., Lu, C., & Foerster, J. (2025). Kinetix: Investigating the training of general agents through open-ended physics-based control tasks. In *The thirteenth international conference on learning representations*. Retrieved from https://arxiv.org/abs/2410.23208

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

Meretzky, S. (1989). *Learning ZIL: Everything you always wanted to know about writing interactive fiction but couldn't find anyone still working here to ask*. Retrieved from https://archive.org/details/Learning_ZIL_Steven_Eric_Meretzky_1995

Nelson, G. (1993). *Inform*. Retrieved from https://github.com/ganelson/inform

Netanyahu, A., Shu, T., Katz, B., Barbu, A., & Tenenbaum, J. B. (2021). Phase: Physically-grounded abstract social events for machine social perception. In *35th aaai conference on artificial intelligence (aaai)*.

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, *15*(12), 1053–1058.

Quilty-Dunn, J., Porot, N., & Mandelbaum, E. (2023). The best game in town: The reemergence of the language-of-thought hypothesis across the cognitive sciences. *Behavioral and Brain Sciences*, *46*, e261.

Schaul, T. (2013). A video game description language for model-based or interactive learning. In *2013 ieee conference on computational inteligence in games (cig)* (pp. 1–8).

Shu, T., Bhandwaldar, A., Gan, C., Smith, K., Liu, S., Gutfreund, D., ... Ullman, T. (2021). Agent: A benchmark for core psychological reasoning. In *International conference on machine learning* (pp. 9614–9625).

Spelke, E. (2022). *What babies know: Core knowledge and composition* (Vol. 1). Oxford University Press.

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

Ullman, T., Baker, C., Macindoe, O., Evans, O., Goodman, N., & Tenenbaum, J. (2009). Help or hinder: Bayesian models of social goal inference. *Advances in Neural Information Processing Systems*, *22*.

Ullman, T. D., Spelke, E., Battaglia, P., & Tenenbaum, J. B. (2017). Mind games: Game engines as an architecture for intuitive physics. *Trends in cognitive sciences*, *21*(9), 649–665.

Wellman, H. M., & Gelman, S. A. (1992). Cognitive development: foundational theories of core domains. *Annual review of psychology*.

Wong, L., Collins, K. M., Ying, L., Zhang, C. E., Weller, A., Gerstenberg, T., ... others (2025). Modeling open-world cognition as on-demand synthesis of probabilistic models. *arXiv preprint arXiv:2507.12547*.