A Case Study in Computational Math Cognition and Embodied Arithmetic

Albert Goldfain (ag33@cse.buffalo.edu)
Department of Computer Science and Engineering,
Center for Cognitive Science,
University at Buffalo, Buffalo, NY 14260

Abstract

This paper defends the claim that a part of human mathematical cognition can be realized in a computational cognitive agent (thereby rejecting the claim that the human embodiment has a privileged status with respect to mathematical cognition). The defense of this position rests on two points: (1) The embodied activities that provide a semantics for arithmetic are performable by computational agents and (2) abstraction from these activities yields representations that can be reasoned with by such agents (while still remaining associated with the originating embodied activities). In particular, such representations can be used in a justification of arithmetic results. A case study in computational math cognition is presented using the task of finding the greatest common divisor (GCD) of two natural numbers. An overview of a preliminary agent implementation is given using the SNePS knowledge-representation, reasoning, and acting system and GLAIR architecture.

Keywords: knowledge representation; cognitive architecture; cognitive modeling

Introduction

Mathematical cognition is an interdisciplinary investigation into the cognitive mechanisms underlying human mathematical activity. This emerging branch of cognitive science has seen contributions from psychology (Piaget, 1965; Gelman & Gallistel, 1978), linguistics (Wiese, 2003), and neuroscience (Dehaene, 1997). Lakoff and Núñez (2000) have provided one of the seminal unifying works in the field of math cognition. The central claim of this work is that mathematics is not objective and disembodied, but rather is subjective and highly dependent on the embodied human mind, along with its particular conceptual system, inference mechanisms, and representations. The possibility of realizing human mathematical cognition in a non-human embodiment (e.g., a computer) is dismissed as being a part of the "Romance of Mathematics" that Lakoff and Núñez hope to dispel. In their view, a computational agent's mechanical embodiment and rigid representations prevent the kind of experiences and conceptualizations that give rise to human mathematical cognition.

In this paper, I argue that at least a part of (developmentally) early mathematical cognition *can* be realized in a computational cognitive agent. By this I mean that such an agent would be capable of passing an empirical test for mathematical *understanding*. Mathematical understanding is a notion I will make more precise and distinguish from mathematical *ability*. As a case study, I look at the problem of finding the GCD of two natural numbers and consider how a cognitive agent (either human or computational) would justify its understanding of GCD.

Along the way, the following semantic questions must be considered and at least partially answered: (1) Where do the semantics for the natural numbers and quantities come from? (2) Where do the semantics for the arithmetic operations come from? (3) Where do the semantics for post-arithmetic operations come from? A theory of math cognition can be formulated that is both cognitively plausible and computational, i.e., suitable for an symbolic-AI agent implementation. The preliminary implementation of such an agent using the SNePS system and the GLAIR architecture is described below. First, we will look at some features of a computational theory of math cognition.

Computational Math Cognition

This work is part of a larger project whose long-term goal is a computational theory of math cognition with a particular emphasis on counting and arithmetic routines. The focus of the theory is on mathematical understanding rather than concept acquisition, learning, or teaching (although these are inextricably tied to understanding).

Ability and Understanding

It will first be necessary to make a sharp distinction between mathematical ability and mathematical understanding. It is fairly uncontroversial to claim that computers (even the unsophisticated pocket calculator) have mathematical ability. Simply put, they are able to produce correct results reliably. Even though it is left to the human user to interpret these results, it is clear that the computer is the one doing the "work" (and thus the request from teachers to "show your work instead of using a calculator"). In contrast, it is not at all clear what the necessary and sufficient conditions for mathematical understanding are or what it would mean for a computational agent to have mathematical understanding. The term 'understanding' is both vague and ambiguous.

Roughly, I take understanding to be a process that happens alongside and after the learning of procedures and acquisition of concepts. From a methodological standpoint, the best we can do is try to identify empirically testable features of mathematical understanding and then look for these features in both human and computational agents. Treated this way, understanding is a gradient of features. A cognitive agent with more of these features can be said to have more understanding than one with less.

The feature of mathematical understanding that I will be focusing on is the ability to justify mathematical results in a common-sense explanation. Although a logic will be used, this is opposed to a formal proof-as-explanation. This will constitute a sort of mathematical Turing test for a computational agent.

Towards a Theory

This section summarizes four important claims from existing research on human mathematical understanding that inform a computational theory.

The representation of numbers and quantities must be flexible. To be cognitively plausible in a symbolic AI system, numbers should be associable with various numeral systems (e.g., Arabic numerals, Roman numerals, talleys) and naming systems (e.g., the number names in some natural language). The representation of number must also allow for the use of numbers as cardinal quantities (e.g., "three apples"), ordinal positions (e.g., "the third apple from the right"), and nominals (e.g., "House number 3") (Wiese, 2003). Quantities must also be able to present themselves in various sensory modalities,

The representation of procedures must be flexible. Mathematical routines can be viewed structurally or operationally (Sfard, 1991). An agent should be able to consider both a given operation and the result of that operation. There is an interplay between procedural and declarative knowledge in mathematical reasoning.

Embodied acts are a source for meaning in early arithmetic Actions involving object collection, object construction, and motion along a path serve as source domains for conceptual metaphor mappings to the abstract objects and operations of arithmetic (Lakoff & Núñez, 2000). So addition, for example, can be conceived of as accumulating two collections of objects, constructing a new object from two pieces, or walking for a distance to a given point and then walking further to another point. Beyond addition, removal can be construed as an implementation of subtraction, iteration as an implementation of multiplication, and sharing as an implementation of division. According to Lakoff and Núñez, these grounding metaphors are combined with further metaphoric mappings (called linking metaphors) and the mapped concepts become applicable to mathematics beyond arithmetic. Mac Lane (1981) speculates that other "originating" human activities serve as the basis for the various branches in higher mathematics.

Mathematics often involves extended cognition. This claim is consistent with observations made by Hutchins (1995). An agent must have the ability to act externally and the ability to represent information beyond the boundary of its embodiment. This includes using external tools to represent this information (e.g., marks on paper, results on a calculator). For a computational agent, this amounts to providing access to external resources and providing a representation scheme for external information that is compatible with the framework already in place for internal information.

Case Study: GCD

I focus on the task of finding the greatest common divisor (GCD) of two natural numbers as a representative case study in computational math cognition. There are several merits to choosing this task. Finding the GCD is an "intermediate-level" problem for a cognitive agent. Its solution requires the simpler arithmetic concepts and operations (or else the geometric analog of arithmetic) and thus can be a showcase for these concepts and operations. However, a solution to the GCD problem need not include the advanced conceptualizations of algebra or calculus. Also, there are many ways to find the GCD of two numbers (perhaps a characteristic of all intermediate level problems). Intuitiveness, efficiency, and applicability will vary greatly from one GCD algorithm to the next.

When asked to find the GCD of two natural numbers *x* and *y* I have found that most non-mathematicians who learned GCD at some point come up with a "back-of-the-envelope" algorithm similar to the following:

```
1. List the divisors of x
2. List the divisors of y
3. List the divisors common to the two lists
above
4. Circle the greatest number from the list of
common divisors
```

This "common-sense" algorithm isolates the semantics of "greatest" and "common" (terms which occur frequently in natural language) from divisor-finding. I would speculate that most non-mathematicians would respond with a similar algorithm (although I have not subjected this hypothesis to a formal trial) and it would be more surprising to see a non-expert using a recursive version of the Euclidean GCD algorithm:

```
gcd(x,y)
      if y=0 return x
      else return gcd(y, x mod y)
      end if
```

This more efficient and optimized algorithm lacks the intuitive semantics of the common-sense algorithm.

To probe the understanding of a cognitive agent who believes that the GCD of 8 and 6 is 2, we can imagine the following idealized question and answer dialogue unfolding:

Q1: Why is 2 the greatest common divisor of 8 and 6?
A1: 2 is the greatest of the common divisors of 8 and 6.
Q2: Why is 2 a common divisor of 8 and 6?
A2: 2 is a divisor of 8 and 2 is a divisor of 6.
Q3: Why is 2 a divisor of 6?
A3: There is a number that, when multiplied by 2, gives 6, and that number is 3.
Q4: Why is 2 times 3 = 6?
A4: Multiplication is repeated addition: 2 plus 2 is 4; 4 plus 2 is 6
Q5: Why is 2 plus 2 = 4?
A5: When I count from 2 for two numbers I end up at 4
Q6: How do you know that you will end up at 4?
A6: I counted two groups of apples, with 2 apples in each, ending up with 4 total apples.
Q7: What is 2?
A7: It is a number and the greatest common divisor of 8 and 6.
Q8: What is a number?
A8: Some examples are 2,4,6 and 8 . . . It is something that can be

counted, added, multiplied ... and something that can be the result of finding a greatest common divisor

This Turing-Test style idealization is often not reproduced by humans. Again informally, I have found that most people will stop somewhere along the line of explaining a complex act in terms of simpler acts and give "It just is" answers (especially when asked things like "Why is 2 times 3 = 6?"). Nevertheless, I think such a dialogue is what we should strive for from a computational agent because at each point a result is justified in terms of simpler, antecedently understood procedures. Humans tend to pack away the procedural knowledge after an arithmetic operation has been understood at a higher level of abstraction. For example, once the multicolumn addition is learned, it seems unnatural to cite counting as a method of addition (see Sfard (1991) for a discussion of this sort of reification).

The dialogue shows the different sources of meaning used in a mathematical justification. Natural language semantics can be used to address Q1 and Q2. A technique of decomposition into simpler procedures is used for Q3, Q4, and Q5. An empirical embodied activity is cited for Q6. Finally, conceptual-role semantics are applied to answer Q7 and Q8 (for a further discussion of these semantic sources, see (Goldfain, 2006a)).

I now consider a how a SNePS-based computational agent might be designed with the ability to compute GCDs and enough understanding (in my sense of the word) to carry out such a question and answer dialogue.

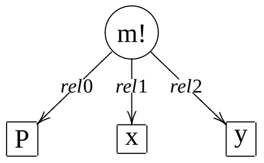
SNePS

SNePS is a knowledge-representation, reasoning, and acting system implemented in Lisp (Shapiro & Rapaport, 1987, 1995). As an agent design platform, SNePS has been used for computational theories of natural-language competence, belief revision, cognitive robotics, and contextual vocabulary acquisition (among many other applications; see (Shapiro, 2007) for a more complete list). As such, SNePS has not been built specifically for mathematical representations or acts, but rather as a general tool for cognitive modeling.

SNePS can function as a frame system, a logical rule-based system, or a propositional semantic network. Corresponding to these representational facets of SNePS are two interface languages for the agent designer: SNePSUL (a Lisp-like language for using SNePS as a semantic network or frame system) and SNePSLOG (a Prolog-like language for using SNePS as a logical rule-based system). For the rest of this paper, I will be expressing SNePS interactions using SNePSLOG, but occasionally showing the semantic network when the relational structure of terms is illustrative. In either interface language, the well-formed formulas (wffs) of the SNePS interface languages represent the intensional beliefs of a cognitive agent named Cassie. A proposition can either be asserted or unasserted in Cassie's belief space. Cassie believes asserted propositions to be true and makes no commitment to the truth value of unasserted propositions. SNePS is *propositional* in that all propositions are represented as first-class terms in the logic. Thus, in the representation of the proposition "3 is the successor of 2", Cassie explicitly represents "3", "2", and the belief "3 is the successor of 2". This an is important feature of SNePS because the agent's beliefs may be the the *object* of further beliefs in the context of an explanation.

The SNePS system includes an inference package (SNIP) that allows Cassie to infer new beliefs through rule-based reasoning (by following explicit rules that she is given) and path-based reasoning (by using relations holding between nodes in her network to infer new relations). SNIP also allows a user to ask Cassie a question by triggering inference. Cassie can handle both true/false/don't know questions and wh-questions (represented by variables to be filled in open propositions). If, in the course of reasoning, Cassie encounters a contradiction between propositions, SNePS has a belief-revision package (SNeBR) that prompts the user to remove one of the contradictory beliefs.

SNePSLOG allows a knowledge engineer to select the relations that structure Cassie's concepts. This is done by defining case-frames with the *define-frame* command. For example, the command `define-frame P(rel0 rel1 rel2)` will make the following case-frame available:



Here, x and y are place-holders. Whenever an assertion is made involving this frame, for example `P(a,b)`, the network structure is built (with `a` and `b` filling in the place-holder slots) and `P(a,b)` becomes an asserted wff in Cassie's belief space.

SNeRE

SNeRE, the SNePS Rational Engine, is Cassie's subsystem for acting. SNeRE provides a set of dedicated case frames for representing acts. There is a distinction made between primitive acts and composite acts. Primitive acts are Cassie's basic repertoire of actions and cannot be further decomposed. Primitive acts are implemented in Lisp. Composite acts are built up out of primitive acts and other acts (both composite and primitive) using SNeRE primitive acts for sequence, conditional, and iteration. I will make use of the following SNeRE primitive acts:

- `ActPlan(a1,a2)`: Act `a1` can be performed by performing plan `a2`. In other words, `a2` is a plan for doing `a1`.
- `believe(p)`: A mental act. The proposition `p` is asserted and added to the list of asserted wffs.
- `disbelieve(p)`: A mental act. The proposition `p` is unasserted and removed from the list of asserted wffs.
- `snsequence(a1,a2)`: The act of first performing `a1` and then performing `a2`

- `withsome(v,p(v),a,d)`: For some `v` such that `p(v)` holds, `a` is performed; if no such `v` can be found, then `d` is performed.

For a complete list of the SNeRE constructs along with their syntax and semantics, see (Shapiro, 2004).

What is most important for our current purposes is that Cassie reasons *while* acting and she leaves a trail of beliefs as an act unfolds. This trail of propositions represents an episodic memory of the particular act that was performed.

GLAIR

GLAIR, the Grounded Layered Architecture with Integrated Reasoning, is an architecture for embodied SNePS agents (Hexmoor & Shapiro, 1997; Shapiro & Ismail, 2003). This architecture has been implemented in various physically embodied agents (robots) and agents with simulated embodiments (softbots). GLAIR is composed of three layers:

- *Knowledge Layer (KL)*: Implemented in SNePSLOG. Contains Cassie's beliefs, plans, and policies.
- *Perceptuo-Motor Layer (PML)*: Implemented in Lisp. Contains Cassie's primitive actions and low-level functions for controlling her embodiment and perception as well as synchronizing percepts with the KL.
- *Sensori-Actuator Layer (SAL)*: Implemented in the particular device implementation language(s) of the selected embodiment. Lowest-level routines for sensing and affecting the outside world. This layer is sometimes omitted or simulated in softbots.

Using GLAIR makes Cassie an agent that can be situated in the real-world or any number of simulated virtual worlds. This allows her to perform concrete embodied activities that will impact her abstract KL beliefs.

Number and Quantity Semantics in SNePS

Numbers for Cassie are positions in a structure exemplifying the natural number progression. All that is required of this structure is a first element and an arbitrary number of unique successive elements ordered by a successor relation. A finite initial segment of this structure must be generated by Cassie during a counting act. The Lisp `gensym` function is used in the PML to produce arbitrarily named unique symbols that become base nodes in the KL (in Figure 1 they are listed as N1, N2, N3 for readability). During counting, each number base node is inserted into the natural-number progression and associated with a numeral. This act of counting numbers differs from counting something *with* numbers. Numbers must participate in quantities, which I take to be a number associated with a sortal. I make an ontological distinction between non-unit sortals like "three apples" and unit sortals "three inches". Non-unit sortals like apples are to be found in object collection activities whereas unit sortals are more useful for object construction and measurement acts. Quantities are represented in SNePSLOG using the `Collection` or `Measure` and `NumSort` case-frames with the following semantics:

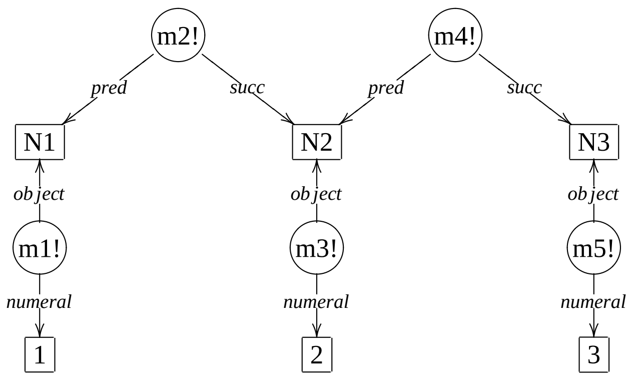
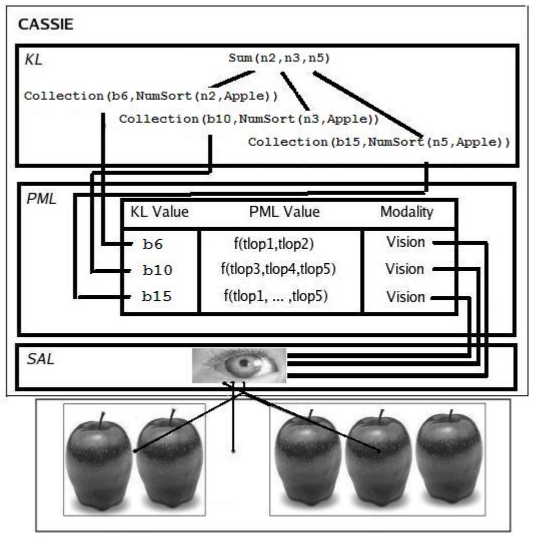


Figure 1: Associating numbers with a syntactic public language (e.g., numerals). ⟦m1⟧ = numeral 1 represents ⟦N1⟧. ⟦m2⟧ = *N*2 is the successor of *N*1. ⟦m3⟧ = numeral 2 represents ⟦N2⟧. ⟦m4⟧ = *N*3 is the successor of *N*2. ⟦m5⟧ = numeral 3 represents ⟦N3⟧.

- `Collection(B,NumSort(N,P))`: B is a collection of N Ps.
- `Measure(B,NumSort(N,P))`: B is an object that measures N Ps.

B is a KL base node associated with a perceptual value in the PML alignment table (see Figure 2). While not a complete solution to the symbol-grounding problem, this is a form of anchoring (Shapiro & Ismail, 2001; Goldfain, 2006b) that allows Cassie to reason about external quantities.


CASSIE
KL
Sum(n2,n3,n5)
Collection(b6,NumSort(n2,Apple))
Collection(b10,NumSort(n3,Apple))
Collection(b15,NumSort(n5,Apple))
PML
SAL

KL Value	PML Value	Modality
b6	f(tlop1,tlop2)	Vision
b10	f(tlop3,tlop4,tlop5)	Vision
b15	f(tlop1, ... ,tlop5)	Vision

Figure 2: Anchoring of quantities in GLAIR. KL symbols (such as `b6`) are aligned to perceptual values (such as top-left-object pixels) in the PML alignment table

Numbers must also play various roles in abstract arithmetic. For example, the number expressed by the numeral 2 can be considered as the successor of 1, the quotient of 8 and 4, or the difference of 3 and 1. Thinking of 2 in these various roles will be dictated by the acts Cassie has performed. This is captured in SNePSLOG using result frames with the following semantics (e.g., for a binary operation):

- `R(arg1 arg2 result)`: The result named `R` of `arg1` and `arg2` is `result`.

So, for example, `Sum(N2,N3,N5)` expresses that the result expressed by "sum" of N2 and N3 is N5. The result name must also be abstracted, because the result named `Sum` may also be expressed by words like `Total` or `Accumulation`.

Embodied Arithmetic

One of the immediate benefits of the GLAIR architecture for embodied arithmetic is that abstract mathematical entities like numbers can be associated with real world objects (like collections) in the form of anchored quantities. This allows various implementations of perceptual routines in the PML to remain independent of the KL logic and reasoning. For example, there is some debate in mathematical cognition as to whether enumeration is performed through the visual modality using object recognition along with a counting routine or whether pattern recognition is used through a subitization routine (e.g., as in the automatic recognition of what is rolled on a die without having to count the dots). Using GLAIR, either perceptual routine could be implemented to work with the vision modality without impacting the "conscious" (i.e., agent-aware) KL reasoning used in a justification. An object-collection implementation is described in detail in (Goldfain, 2006b). This activity is sufficient for answering questions like Q6 in the dialogue above.

Recursive (Disembodied) Arithmetic

The recursive notion of understanding demonstrated in the idealized interrogation above suggests that a new act is understood in terms of previously understood acts. Mathematics in general and arithmetic in particular are very prerequisite-dependent subjects of thought. This creates a continuum of understanding (Rapaport, 1995) that links up procedural knowledge (how can I do this with things I already know how to do?) and conceptual knowledge (how do the results of my actions relate to one another?).

Figure 3 shows how multiplication as iterated (strictly speaking recursive) addition can be expressed in SNePSLOG, a procedure I am calling add-multiplication. The procedural link between addition and multiplication is also augmented by two important rules:

```
all(x,y)({Number(x),Number(y)} &=>
      ActPlan(Multiply(x,y),AddMultiply(x,y))).
all(x,y,z)(SumProduct(x,y,z) => Product(x,y,z)).
```

The first rule says that one way of multiplying two numbers is add-multiplication. In fact Cassie might know several ways of performing the act `Multiply(x,y)` and may decide to use another procedure for multiplication outside the context of explanation (see (Shapiro *et al.* (2007))) for further discussion). The second rule ties together the result frames from the specific act of multiplication and the general act of multiplication by saying that a summed-product is a product *simpliciter*. The general act result frame allows Cassie to pull together results obtained from different sources under a common representation. Thus, arithmetic can be expressed recursively for Cassie with a basis of the counting act. Addition is counting from the first addend for the-second-addend-many-numbers (a sort of metacounting). Subtraction is inverted addition (i.e., finding a number that, when added to the number being subtracted, will yield the number being subtracted from). Multiplication is iterated addition as shown. Division is inverted multiplication.

GCD-Specific Acts

What remains is to account for how the arithmetic acts are linked to the GCD specific acts of building divisor lists, building a common list, and picking out the greatest element of a list.

The building of the divisor lists is done by repeatedly trying successive "candidate" divisors along the number line for each input argument to GCD. Cassie does not need a notion of a remainder for division since she only knows about the natural numbers (and thus she will only infer whole number answers to division problems). A list of common divisors is constructed by using the logical `and` connective provided by SNePSLOG to have Cassie pick out elements that are in both lists. These lists are built in the KL but represented in a Lisp "scratch pad", a form of extended cognition so that Cassie does not have to keep the lists "in her head". The semantics from "greatest" are given by path-based reasoning. If a number `y` can be reached using a path consisting of inverse `pred` arcs and forward `succ` arcs (see Figure 1) from a number `x`, it must be the case that $x < y$. Basically, this represents Cassie's implicit knowledge that the numbers said later in the counting act are greater than those said earlier.

Conclusion

I have shown how a series of mathematical acts can be implemented in a computational cognitive agent. In principle, the effects of these acts (represented by result-frames in SNePS) are all that an agent would require to carry out a Turing-test style demonstration of understanding. I believe that those who, like Lakoff and Núñez, maintain that computational agents cannot attain mathematical understanding must specify some empirical test by which a feature of mathematical understanding can be measured in humans and demonstrate why a computational agent would fail such a test.

Acknowledgments

I am grateful to William J. Rapaport, Stuart C. Shapiro, four anonymous reviewers and the members of the SNePS Research Group for commenting on previous versions of this work.

```
all(x)(ProductResultSoFar(x) =>
       ActPlan(AddFromFor(x,0),believe(SumProduct(x,1,x)))).
all(w,x,y,z)({NumToAdd(w), Successor(y,z),ProductResultSoFar(x)} &=>
             {ActPlan(AddFromFor(x,y),
                      snsequence(disbelieve(ProductResultSoFar(x)),
                      snsequence(Add(x,w),
                                 withsome(?p,
                                          Sum(x,w,?p),
                                          snsequence(believe(ProductResultSoFar(?p)),
                                                     AddFromFor(?p,z)),
                                          Say(``I don't know the sum''))))))}).
all(x,y)({Number(x),Number(y)} &=>
          ActPlan(AddMultiply(x,y),
                  snsequence(believe(ProductResultSoFar(x)),
                  snsequence(believe(NumToAdd(x)),
                  snsequence(withsome(?p,Successor(?p,y),AddFromFor(x,?p),AddFromFor(x,0)),
                  snsequence(withsome(?z,ProductResultSoFar(?z),
                                      snsequence(believe(SumProduct(x,y,?z)),
                                                 disbelieve(ProductResultSoFar(?z))),
                                      SayLine("I could not determine the product.")),
                             disbelieve(NumToAdd(x)))))))).
```

Figure 3: Add Multiplication in SNePS. The recursive and base cases of the act `AddFromFor` and the plan for `AddMultiply`

References

Dehaene, S. (1997). *The Number Sense*. Oxford, UK: Oxford University Press.

Gelman, R., & Gallistel, C. R. (1978). *The Child's Understanding of Number*. Cambridge: Harvard University Press.

Goldfain, A. (2006a). A Computational Theory of Inference for Arithmetic Explanation. *Proceedings of ICoS-5*, 145–150.

Goldfain, A. (2006b). Embodied Enumeration: Appealing to Activities for Mathematical Explanation. *Cognitive Robotics: Papers from CogRob2006*, 69–76.

Hexmoor, H., & Shapiro, S. C. (1997). Integrating Skill and Knowledge in Expert Agents. In P. J. Feltovich and K. M. Ford and R. R. Hoffman (Ed.), *Expertise in Context* (p. 383-404). Menlo Park, CA / Cambridge, MA: AAAI/MIT.

Hutchins, E. (1995). *Cognition in the Wild*. Cambridge, MA: MIT Press.

Lakoff, G., & Núñez, R. (2000). *Where Mathematics Comes From*. New York: Basic Books.

Mac Lane, S. (1981). Mathematical Models. *American Mathematical Monthly*, *88*, 462–472.

Piaget, J. (1965). *The Child's Conception of Number*. New York, NY: Norton.

Rapaport, W. J. (1995). Understanding Understanding. In J. Tomberlin (Ed.), *AI, Connectionism, and Philosophical Psychology* (p. 49-88). Atascadero: Ridgeview.

Sfard, A. (1991). On the Dual Nature of Mathematical Conceptions. *Educational Studies in Mathematics*, *22*, 1–36.

Shapiro, S. C. (2004). *SNePS 2.6.1 User's Manual*. http://www.cse.buffalo.edu/sneps/.

Shapiro, S. C. (2007). *The SNePS Research Group Bibliography*. http://www.cse.buffalo.edu/sneps/Bibliography/.

Shapiro, S. C., & Ismail, H. O. (2001). Symbol-Anchoring in Cassie. In S. Coradeschi & A. Saffioti (Eds.), *Anchoring Symbols to Sensor Data in Single and Multiple Robot Systems* (p. 2-8). AAAI Press.

Shapiro, S. C., & Ismail, H. O. (2003). Anchoring in a Grounded Layered Architecture with Integrated Reasoning. *Robotics and Autonomous Systems*, *43*, 97–108.

Shapiro, S. C., & Rapaport, W. J. (1987). SNePS Considered as a Fully Intensional Propositional Semantic Network. In N. Cercone & G. McCalla (Eds.), *The Knowledge Frontier* (p. 262-315). New York, NY: Springer Verlag.

Shapiro, S. C., & Rapaport, W. J. (1995). An Introduction to a Computational Reader of Narrative. In J. F. Duchan, G. A. Bruder, & L. E. Hewitt (Eds.), *Deixis in Narrative* (p. 79-105). Hillsdale, NJ: Lawrence Erlbaum.

Shapiro, S. C., Rapaport, W. J., Kandefer, M., Johnson, F. L., & Goldfain, A. (2007). Metacognition in SNePS. *AI Magazine*, *28*(1), 17–31.

Wiese, H. (2003). *Numbers, Language, and the Human Mind*. Cambridge, UK: Cambridge University Press.