# Bridging the Gap Between Schank and Montague

**John F. Sowa and Arun K. Majumdar**
VivoMind Research, LLC
sowa@vivomind.com arun@vivomind.com

**Abstract**

Documents that people write to communicate with other people are rarely as precise as a formal logic. Yet people can read those documents and relate them to formal notations for science, mathematics, and computer programming. They can derive whatever information they need, reason about it, and apply it at an appropriate level of precision. Unlike theorem provers, people rely on analogies for their reasoning. Even mathematicians use analogies to discover their theorems and formal proofs to verify and codify their discoveries. This article shows how a high-speed analogy engine is used to analyze natural language texts and relate the results to both structured and unstructured representations. The degree of precision in the results depends more on the precision in the knowledge sources used to analyze the documents than on the precision of the language in the documents themselves.

## 1. Neat and Scruffy Methods

Commonsense reasoning is based on the way people think and talk. Human reasoning is closely related to perception and mental models, and language relates words to those models by analogies and metaphors. Formal logic and ontology, however, are abstractions from language that replace perceptual patterns with symbolic expressions. Computers can manipulate symbols much faster and more accurately than humans, but they're not as good at analogical reasoning. The most challenging task for relating language to commonsense reasoning is to develop methods of analogy for relating patterns of words to patterns of reasoning.

For natural language processing, Roger Schank represents scruffy methods for matching language patterns to domain-dependent background knowledge. Richard Montague (1970b) represents the neat extreme of treating language as a formal system: "There is in my opinion no important theoretical difference between natural languages and the artificial languages of logicians; indeed, I consider it possible to comprehend the syntax and semantics of both kinds of languages within a single natural and mathematically precise theory." At that level, Schank and Montague are irreconcilable. Montague is the kind of logician that Schank denounced as misguided or at best irrelevant. Their only point of agreement is their opposition to Chomsky and "the developments emanating from the Massachusetts Institute of Technology" (Montague 1970a). Yet in their reaction against Chomsky, both Montague and Schank evolved positions that are remarkably similar, although their notations hide the resemblance. What Chomsky called a noun, Schank called a picture producer, and Montague called a function from entities to truth values. But Schank never produced any pictures, and Montague never applied his functions to any entity.

Neat and scruffy methods can be compared in common terms: semantics, not syntax, is key to understanding language. The traditional grammatical categories are surface manifestations of underlying semantic categories. Each word has a characteristic semantic pattern — a lambda expression for Montague or a graph for Schank — that determines how it combines with other words. Grammar helps to guide the semantics in constructing a representation for each phrase and sentence of discourse. The variety of complex sentences is not the result of a complex grammar, but of the interactions between a simple grammar and the variety of semantic patterns. Finally, the denotation of a sentence is determined by relating its semantic representation to a model of the domain. Schank never talked about truth values or denotations, but his programs computed them.

The most significant difference between Schank and Montague is in their use of background knowledge. Montague adhered to Frege's principle of *compositionality*, which claims that the meaning of a sentence is derived from the meaning of the words it contains and the grammar rules for combining words. Montague's lexicon contains

all the knowledge used in language analysis: each word is defined by one or more logical expressions, and each grammar rule has an associated semantic rule for combining those expressions. Schank (1982) included some semantics in the lexicon, but he organized most of the background knowledge in domain-dependent *scripts*, *memory organization packets* (MOPs), and *theme organization packets* (TOPs). With Schank's methods, background knowledge is more important for semantic interpretation than the information in the lexicon.

Background knowledge can be processed at any level of precision from neat to scruffy, but the most challenging problem is to find the relevant knowledge when it is needed. To make background knowledge easily accessible at parse time, word-expert parsers (WEP) store all the information in a huge lexicon indexed by the word forms (Small 1980). But the amount of knowledge is open ended, and WEP lexicons must encode and store information about every specialized word sense. Formal ontologies and knowledge bases are usually organized by domain, and they may require a parser to do a considerable amount of analysis before it can find the relevant information. Statistical parsers could be trained to associate patterns of words to some semantic notation, but they would require human experts to annotate a training corpus for every domain and genre. A statistical parser trained on a corpus of well-edited documents cannot handle ungrammatical or fragmentary text about the same domain.

The VivoMind technology uses novel methods to address these issues: conceptual graphs for a semantic representation that spans a range of precision from formal logic to the scruffiest heuristics; a high-speed analogy engine for finding relevant graphs in log(N) time, where N is the number of graphs in the knowledge base; methods of language analysis and reasoning that can use knowledge in structured or unstructured forms; and semi-automated tools that enable subject-matter experts to extend the knowledge base without having to learn special notations or methods for knowledge acquisition. Section 2 introduces the VivoMind Analogy Engine (VAE), its use in analogical reasoning, and its support for other methods of reasoning by finding relevant information upon request. Section 3 discusses the use of VAE and the VivoMind Language Processor (VLP) in three applications, which have different tolerances on precision in the knowledge representation and reasoning. The concluding Section 4 discusses some implications of this approach for linguistic theory and practical applications.

## 2. Analogy Engine

Analogies have been an important area of research since the early days of AI, but they are often studied as an aspect of some other topic, such as machine learning, case-based reasoning, metaphor, or just pattern matching. The goal of pattern matching is the recognition and mapping of related parts of two structures. Analogy finding adds the goal of searching for analogous structures in a knowledge base of arbitrary size. Both neat and scruffy methods of reasoning and language analysis use repeated steps of pattern matching and analogy finding. The main difference between them is in the constraints on permissible matches. The three methods of formal reasoning can be considered special cases of analogy:

- **Deduction.** A typical rule used in deduction is *modus ponens*: given an assertion *p* and an axiom of the form *p* implies *q*, deduce the conclusion *q*. In most applications, the assertion *p* is not identical to the *p* in the axiom, and structure mapping is necessary to *unify* the two *p*s before the rule can be applied. The most time-consuming task is not the application of a single rule, but the repeated use of analogies for finding patterns that may lead to successful rule applications.
- **Induction.** When every instance of *p* is followed by an instance of *q*, induction is performed by assuming that *p* implies *q*. Since the *p*s and *q*s are rarely identical in every occurrence, a form of analogy called *generalization* is used to derive the most general implication that subsumes all the instances.
- **Abduction.** The operation of guessing or forming an initial hypothesis is what Peirce called abduction. Given an assertion *q* and an axiom of the form *p* implies *q*, the guess that *p* is a likely cause or explanation for *q* is an act of abduction. The operation of guessing *p* uses the least constrained version of analogy, in which some parts of the matching graphs may be more generalized while other parts are more specialized.

With appropriate constraints on the permissible pattern matching, a general-purpose analogy engine can perform any combination of informal analogies or formal steps of deduction, induction, and abduction. For VivoMind software, conceptual graphs (CGs) are the primary knowledge representation, and the VivoMind Analogy Engine (VAE) serves as a high-speed associative memory for CGs (Sowa & Majumdar 2003). At the neat extreme, the logical structure of CGs is based on the existential graphs by Peirce (1906, 1909), they have the model-theoretic semantics of Common Logic (ISO/IEC 24707), and VAE can find matching graphs that satisfy the strict constraints of unification. At the scruffy extreme, CGs can represent Schank's conceptual dependencies, scripts, MOPs, and TOPs. VAE can support case-based reasoning (Schank 1982) or any heuristics used with semantic networks. The wide range of reasoning methods is a good reason for using CGs. A more important reason is the graph structure, which allows the use of mathematical theories, encodings, and algorithms that are rarely exploited in AI.

The Structure-Mapping Engine (SME) pioneered a wide range of methods for using analogies (Falkenhainer et al. 1989; Lovett et al. 2010). But SME takes N-cubed time to find analogies in a knowledge base with N options. For better performance, conventional search engines can reduce the options, but they are based on an unordered bag of words or other labels. Methods that ignore the graph structure cannot find graphs with similar structure but different labels, and they find too many graphs with the same labels in different structures.

Organic chemists developed some of the fastest algorithms for representing large labeled graphs and efficiently finding graphs with similar structure and labels. Chemical graphs have fewer types of labels and links than conceptual graphs, but they have many similarities. Among them are frequently occurring subgraphs, such as a benzene ring or a methyl group, which can be defined and encoded as single types. Algorithms designed for chemical graphs (Levinson & Ellis 1992) were used in the first high-speed method for encoding, storing, and retrieving CGs in a generalization hierarchy. More recent algorithms encode and store millions of chemical graphs in a database and find similar graphs in logarithmic time (Rhodes et al. 2007). By using a measure of graph similarity and locality-sensitive hashing, their software can retrieve a set of similar graphs with each search.

Chemical graphs have a regular structure that enables systematic algorithms to encode the graphs and operate on them. By comparison, predicate calculus has a much more complex syntax. The mapping by Gödel (1931), for example, requires a large number of irregular functions to operate on the encoding. Peirce's existential graphs (EGs) have the simplest syntax for logic ever invented, and they are capable of expressing the full semantics of Common Logic (Sowa 2009). Conceptual graphs are a typed version of EGs, the typing helps reduce the number of nodes in the graphs, and the canonical formation rules define a CG grammar that is tied to Peirce's rules of inference. In effect, a CG with no negations has the same structure as a chemical graph. Negations extend CGs to nested structures in which some nodes contain other CGs.

The original version of VAE used algorithms related to those for chemical graphs. More recent variations have led to a family of algorithms that encode a graph in time that is polynomial in the size of a graph and store or retrieve an encoding in log(N) time, where N is the total number of graphs. With a semantic distance measure based on both the structure of the graphs and an ontology of their labels, locality-sensitive hashing can retrieve a set of similar graphs in log(N) time. With this speed, VAE can find analogies in a knowledge base of any size without requiring a search engine as a preliminary filter. The next section describes how it is used in reasoning and language analysis..

## 3. VivoMind Language Processor

The VivoMind Language Processor (VLP) is a semantics-based language interpreter that translates natural language text to conceptual graphs (Majumdar et al. 2008, 2009). For syntax, VLP uses a link grammar with labels that correspond to thematic roles, such as Agent, Patient, Theme, Experiencer, Recipient, Instrument, Result, and Attribute. All semantic patterns are represented as CGs. Some of them, called *canonical graphs*, are associated with words in the lexicon, and their relations have the same labels as the links in the grammar. Other CGs are retrieved from background knowledge by VAE, and they may use any concept and relation types appropriate to the subject matter. Although VLP is based on link grammar, its control structure is strongly influenced by a distributed concurrent dependency parser called ParseTalk (Hahn et al. 1994, 2000). Like ParseTalk, VLP uses a message-passing protocol that allows multiple agents to perform arbitrary semantic and pragmatic processing during the syntactic analysis.

The interpretation generated by VLP is a well-formed CG, but some of the concepts and relations may be underspecified. Noun-noun combinations, for example, provide no syntactic clues to the semantics: a steamer clam is a type of clam that is cooked by steam, but a steamer duck is a type of duck that flaps its wings like a paddle-wheel steamer. If the combination is not stored in the lexicon, the concepts generated from the nouns are connected by the unspecified relation **(Link)**. To specialize that vague link to a more detailed relation or subgraph, VAE retrieves background knowledge, which may be CGs derived from any source: a knowledge base, a database, or previously analyzed text in the same document or other related documents.

To illustrate the use of VAE in language analysis and the level of precision in the reasoning, following are three applications processed by a language analyzer coupled with VAE. The first two used an older analyzer called Intellitex, and the third used VLP:

1. **Evaluating student answers**. A textbook publisher wanted software to grade student solutions to word problems in mathematics. But the solutions were written in free-form English sentences. They were too short for statistical measures to be reliable, and they were too ungrammatical and unsystematic to be translated to logic and be verified by a theorem prover.

2. **Legacy re-engineering**. A large corporation had software in daily use that was up to 40 years old. It included about 1.5 million lines of COBOL programs, which were documented by 100 megabytes of English text in reports, manuals, email notes, and comments written in the programs. The task was to analyze and compare different versions of the software and different versions of the documentation. The goal was to detect and report discrepancies between the programs and the

documentation, to compile a glossary of all the terminology used in the documents with cross references to the software, and to create data structure diagrams and process diagrams of the programs, files, and data.

3. **Oil and gas exploration**. The task was to analyze 79 documents about oil and gas fields, which included site reports about fields around the world and chapters from a textbook on geology that could be used for background information. The goal was to analyze new site reports, compare them to the previously analyzed reports, determine which ones were the most closely related, and produce a detailed comparison of similarities and differences between the new site and any of the previously analyzed sites.

For the task of evaluating student answers, the CG in Figure 1 is a canonical graph associated with the verb *multiply*. The boxes represent concepts, the ovals represent relations, and the diamond represents a function. The boxes and ovals can be matched to a variety of sentence patterns about somebody multiplying a number by a number to produce a product. For any sentence they match, the diamond is carried along to show that the product is a function of the two numbers. Similar canonical graphs would represent ways of talking about other mathematical operators. The result of interpreting a mathematical description would be a join of several such CGs on their common concepts. The attached diamond nodes would form a dataflow graph that represents an arithmetic expression.

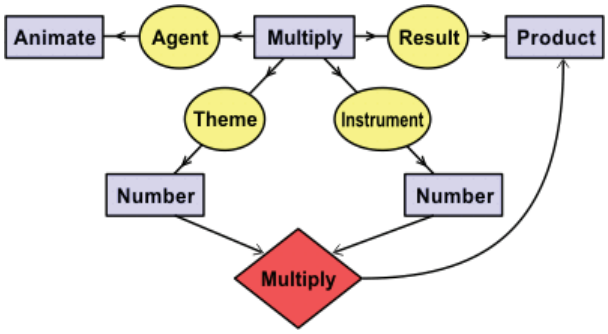


**Fig 1. A conceptual graph for interpreting sentences about multiplication**

The *display form* of CGs illustrated in Figure 1 is useful for readability. The Conceptual Graph Interchange Format (CGIF), which is one of the three normative dialects in the Common Logic standard, is a serialization of the display form that represents concept nodes in square brackets and relation and function nodes in parentheses. The display form and CGIF are logically, but not structurally equivalent to the Common Logic Interchange Format (CLIF). Following is the CLIF version of Figure 1:

```
(exists (x1 Animate) (x2 Multiply) (x3 Product)
   (x4 Number) (x5 Number))
   (and (Agent x2 x1) (Result x2 x3) (Theme x2 x4)
     (Instrument x2 x5) (= x3 (Multiply x4 x5)) ))
```

For each of the word problems, the publisher had already collected about 50 student solutions from previous exams, and each one had an evaluation written by some teacher who said that the solution was correct, incorrect, or partially correct. For each partially correct solution, the teacher added a comment that said what was missing. The VivoMind approach to the publisher's task was to use Intellitex, VAE, and the canonical graphs for arithmetic to translate the student solutions to conceptual graphs. Then the task of evaluating solutions by other students could be handled by case-based reasoning: The new solution was translated to a conceptual graph, which VAE compared to the 50 previous cases, and the semantic distance measure determined which was the best match. If that match was within an acceptable tolerance, the teacher's evaluation for the matching solution was selected. If VAE couldn't find any acceptable match, the new solution could be sent to some teacher for evaluation; that solution-evaluation pair would then be added to the set of cases in order to improve the coverage. For all the examples tested, this method selected evaluations that were considered appropriate.

For the legacy re-engineering task, translating English to COBOL is impossible. But COBOL is a formal language, off-the-shelf grammars are available for parsing it, and the COBOL parse trees can easily be translated to conceptual graphs. Those CGs can then be used as the knowledge base for interpreting the English documentation. Any sentences that don't mention anything about the COBOL programs are ignored as irrelevant. Those sentences that do mention something in the programs are matched to the CGs derived from those programs. The pattern matching is similar to the process of matching Figure 1 to a sentence about multiplication. Information from one source can fill gaps in graphs derived from the other source, but conflicts and constraint violations are noted. The project was completed successfully, and all the information the client had requested was generated and written on a CD-ROM.

For oil and gas exploration, the task of comparing a report about a new site to the reports about previous sites is more complex than just measuring some semantic distance between the reports. Instead of a single number, the geologists wanted a side-by-side comparison of similar and contrasting features. Furthermore, the words used to describe similar features might not be identical, and the words for describing contrasting features are almost certainly different. Instead of relating two site reports directly, VAE would typically find chains of concepts and relations that extend from one site report to one or more chapters from a textbook on geology and then to graphs in another site report that are expressed with different concepts. For each site related to the query, VLP finds multiple documents that contained CGs derived from the query, CGs from the site report, and CGs from the textbook or other reports that contain background information. In a sense, VLP "learns" new information by reading a book. But for each query, it focuses only on

those parts of the book that are useful for relating the query to the answer. This method is very different from current IR, IE, and DB systems:

- IR systems typically use a "bag of words" method to measure the similarity of a query to a document that might contain an answer to that query. But they don't extract the information and summarize it in a table or paragraph. It's possible to apply IR methods to individual paragraphs, but that technique would miss documents in which the significant words are scattered in different paragraphs. And no IR systems connect partial information from multiple documents.
- IE systems extract particular pieces of information, and some can link multiple pieces from different documents. Typical IE systems use predefined templates that specify expected syntactic and semantic patterns, but they have stagnated at about 60% accuracy. Hobbs and Riloff (2010) noted "it is not clear what we can do to overcome [that barrier], short of solving the general natural language problem in a way that exploits the implicit relations among the elements of a text." VLP doesn't need predefined templates. CGs derived from the query enable it to find implicit relations in a textbook and exploit them to generate precise answers.
- DB systems can relate and combine information from multiple sources, but they use query languages like SQL and SPARQL. Some support English-like front ends, but all the information they access must be predigested and translated to whatever format the database system requires.

Although conceptual graphs are defined as a formal logic, precise logic cannot be derived from a vague sentence. The CG that represents a sentence is actually derived by combining CGs from previously acquired knowledge. The precision of the result is determined by the precision of the original CGs. This method violates Frege's principle of *compositionality*, which says that the meaning of a sentence is derived from the meaning of the words it contains and the grammar rules for combining words. Montague was a strict adherent: each word is defined by one or more logical expressions, and each grammar rule has an associated semantic rule for combining those expressions. Montague allowed some words to have multiple meanings, but the grammar rules check semantic constraints to determine the correct option in each case. To support context-dependent references, Kamp's DRT uses information outside the sentence to determine interconnections. Both neat and scruffy systems make tradeoffs between the amount of meaning stored in the lexicon and the amount derived from context or general background knowledge. The high-speed analogy engine enables VLP to find and use much more background knowledge than most NLP systems.

In summary, VLP uses a combination of lightweight, middleweight, and heavyweight semantics. For any text, the broad outline of meaning comes from lightweight resources such as WordNet combined with middleweight ontologies with few axioms and definitions. The detail comes from background knowledge represented in conceptual graphs. At the heavyweight extreme, those CGs may be derived from formal logics, programming languages, or highly structured databases. The Common Logic standard (ISO/IEC 2007) specifies a model-theoretic semantics for CGs. But CGs have extensions beyond the CL standard (Sowa 2009), and they can also be used with "scruffy" heuristic methods.

## 4. Implications for Theory and Practice

Different people can derive a different amount of meaning or even a different degree of precision of meaning from the same sentence. Their interpretation of the sentence depends heavily on their general education and their knowledge of the subject matter. The same principle is true of computer systems.

Most systems that translate language to logic generate the logic by combining smaller logical expressions to form larger ones. For systems that obey Frege's principle of compositionality, those smaller expressions represent word meanings, which are usually stored in the lexicon. If the lexicon is static, such systems cannot learn. For systems that depend heavily on background knowledge, some of the meaning may be stored in the lexicon, but more of it may come from domain-dependent resources. For the oil and gas example, VAE would frequently find important contributions in chapters from the geology textbook. For the legacy re-engineering example, the CGs derived from COBOL were a major source of the meaning used to derive the interpretation.

For the legacy re-engineering project, the level of precision of the language analysis and the complexity of the reasoning were quite high. That precision was the result of the large number of CGs derived by translation from a formal language. Even though the English documents varied widely in style and accuracy, the interpretations formed by joining CGs derived from COBOL were always precise.

For the task of evaluating student answers, the subject matter was precise, and the canonical graphs, such as Figure 1, were precise. The student answers, however, varied widely. The A students, as expected, made correct statements, which would produce correct dataflow diagrams when they were interpreted by graphs such as Figure 1. But many students, including some A students, might include some irrelevant verbiage. Variations of the semantic distance measure can place greater weight on graphs and subgraphs whose ontology is critical to the application.

For linguistics, these results are more compatible with the theories of language games by Wittgenstein (1953) or microsenses by Cruse (2000) than with any theory that assumes a fixed ontology with fixed word senses (Sowa 2010). People have a high-speed associative memory, which allows them to take advantage of huge resources of background knowledge for generating and interpreting language. Computer systems with commonsense will also need a high-speed associative memory in order to understand what humans say and write.

## References


Cruse, D. A. 2000. Aspects of the micro-structure of word meanings, in Y. Ravin and C. Leacock, eds., Polysemy: Theoretical and Computational Approaches. Oxford: University Press, pp. 30-51.

Falkenhainer, B., K. D. Forbus, and D. Gentner. 1989. The structure mapping engine: algorithm and examples, *Artificial Intelligence* **41**, 1-63.

Gödel, K. 1931. On formally undecidable propositions of *Principia Mathematica* and related systems, in J. van Heijenoort, *From Frege to Gödel*, Cambridge, MA: Harvard University Press. pp. 592-617.

Hahn, U., S. Schacht, and N. Bröker. 1994. Concurrent natural language parsing: The ParseTalk model. *International Journal of Human-Computer Studies* **41**, 179-222.

Hahn, U., N. Bröker, & P. Neuhaus. 2000. Let's ParseTalk: Message-passing protocols for object-oriented parsing, in H. Bunt and A. Nijholt, eds., *Recent Advances in Parsing Technology*, Dordrecht: Kluwer.

Hobbs, J. R., and E. Riloff. 2010. Information extraction, *Handbook of Natural Language Processing*, 2nd edition, edited by N. Indurkhya and F. J. Damerau, CRC Press.

ISO/IEC. 2007. *Common Logic (CL) – A Framework for a family of Logic-Based Languages*, IS 24707, International Organisation for Standardisation, Geneva.

Levinson, R. A,, and G. Ellis. 1992. Multilevel hierarchical retrieval, *Knowledge Based Systems* **5:3**, pp. 233-244.

Lovett, A., K. Forbus, and J. Usher. 2010. A structure-mapping model of Ravenandrsquo;s Progressive Matrices, *Proceedings of CogSci-10*, pp. 2761-2766.

Majumdar, A. K., J. F. Sowa, and J. Stewart. 2008. Pursuing the goal of language understanding, in P. Eklund LNAI 5113, Berlin: Springer. http://www.jfsowa.com/pubs/pursuing.pdf

Majumdar, A. K., and J. F. Sowa. 2009. Two paradigms are better than one and multiple paradigms are even better, in S. Rudolph, F. Dau, and S.O. Kuznetsov, eds., *Conceptual Structures: Leveraging Semantic Technologies*. Berlin: Springer. http://www.jfsowa.com/pubs/paradigm.pdf

Montague, R. 1970a. English as a formal language, reprinted in Montague. 1974., pp. 188-221.

Montague, R. 1970b. Universal grammar, reprinted in Montague. 1974., pp. 222-246.

Montague, R. 1974. *Formal Philosophy*, Yale University Press, New Haven.

Peirce, C. S. 1906. Manuscripts on existential graphs, *Collected Papers of C. S. Peirce*, vol. 4, Harvard University Press, Cambridge, MA, pp. 320-410.

Peirce, C. S. 1909. Manuscript 514. With commentary by J. F. Sowa. http://www.jfsowa.com/peirce/ms514.htm.

Rhodes, J., S. Boyer, J. Kreulen, Ying Chen, and Patricia Ordonez. 2007. Mining patents using molecular similarity search, *Pacific Symposium on Biocomputing* **12**, 304-315.

Schank, R. C. 1982. *Dynamic Memory*, Cambridge University Press, New York.

Small, S. I. 1980. *Word Expert Parsing*, PhD Thesis, Department of Computer Science, University of Maryland.

Sowa, J. F. 2009. Conceptual Graphs for Conceptual Structures, in P. Hitzler and H. Schärfe, eds., *Conceptual Structures in Practice*, Chapman and Hall/CRC Press, pp. 102-136. http://www.jfsowa.com/pubs/cg4cs.pdf

Sowa, J. F. 2010. The role of logic and ontology in language and reasoning, Chapter 11 of *Theory and Applications of Ontology: Philosophical Perspectives*, edited by R. Poli and J. Seibt, Berlin: Springer, pp. 231-263. http://www.jfsowa.com/pubs/rolelog.pdf

Sowa, J. F., and A. K. Majumdar. 2003. Analogical reasoning, in A. de Moor, W. Lex, and B. Ganter, eds., *Conceptual Structures for Knowledge Creation and Communication*, LNAI 2746, Springer. http://www.jfsowa.com/pubs/analog.htm

Sowa, J. F., and A. K. Majumdar. 2008. Pursuing the goal of language understanding, Slides presented at CSLI, Stanford. http://www.jfsowa.com/talks/pursue.pdf

Wittgenstein, L. 1953. *Philosophical Investigations*, Oxford: Blackwell.