Chapter 5

Language & Representation

Time flies like an arrow
Fruit flies like a banana
Groucho Marx

Natural language is ambiguous. Words have multiple meanings, which can lead to several ways of parsing a sentence. An example used frequently in natural language processing is "Time flies like an arrow", which can mean 1) Time flies the way an arrow does, 2) Measure how fast the flies are, as you would an arrow, 3) Measure how fast the flies are, as an arrow would, or 4) A special kind of fly, the time fly, likes arrows [5-1]. Sentences can also be interpreted literally or figuratively, as in the sentence "She flew to the bank" which may mean she took an airplane to the bank, or she went to the bank quickly. The bank may be for holding money, or it may be the bank of a river.

5.1. Grammar

Natural language allows us to make complex assertions about the world. The database and semantic network *systems* explored in the previous chapter offer various degrees of expressive power and flexibility but are all simplifications of written language. The particular differences between them will be formally investigated here and a new design introduced for representing complex knowledge that is both flexible and efficient.

The relational database is, semantically speaking, the simplest system introduced so far. To observe the assertions made in a relational database, we can "translate" the meaning of the database back into natural language. An interpretation of Figure 4.7. from the previous chapter would give us:

Table 5.1. Translating a relational database back into English language.

Galileo is a person.	Galileo was born in Pisa.
Copernicus is a person.	Copernicus was born in Turin.
Newton is a person.	Newton was born in Woolsthorpe.
Einstein is a person.	Einstein was born in Ulm.
Heisenberg is a person.	Heisbenberg was born in Wurzburg.
Newton was a physicist.	Galileo was an astronomer.
Einstein was a physicist	Copernicus was an astronomer.
	Heisenberg was a physicist.

.

This is a direct translation of the assertions made in the database. If we were to read this in a book, it would be found in the more compact form of natural language:

Galileo and Copernicus, both astronomers, were born in Pisa and Torun respectively. The physicists Newton, Einstein and Heisenberg were born in Woolsthorpe, Ulm and Wurzburg.

The fact that the names refer to people is taken for granted while the reader is assumed to know, or be able to look up, what countries these cities are found in. Interestingly, the natural language version uses twenty-five words while the expanded version above uses sixty-nine. However, when stored in a relational database only twelve words are needed: five people, five places, and two occupations, while the relationships take negligible additional space. These issues of expressiveness and efficiency will be revisited.

Many sentences can be deconstructed in this way. Consider another example from Asimov's encyclopedia of science and technology:

> "Turing attended Cambridge University and was elected a fellow of King's College, Cambridge in 1935. Between 1936 and 1938 Turing worked at Princeton University in New Jersey and, while there, dealt with the theoretical concept of the so-called Turing machine, a computer capable of the most general computations." [5-2]

Which is expanded as:

Turing is a person.
Turing attended Cambridge University.
Turing was elected fellow at King's College in 1935.
Turing worked at Princeton University from 1936 to 1938.
Princeton University is in New Jersey
Turing developed the Turing Machine.
Turing worked on the Turing Machine while at Princeton University.
The Turing Machine is a theoretical computer.
The Turing Machine is capable of general computation.

This would be difficult to represent with a relational database as we must define the schema for people, universities, inventions, geography, machines, and things-that-happen-to-people - ahead of time. How many tables should be defined? The number of possible relationships cannot be known a priori. Thus the relational database, while suitable for certain kinds of knowledge, is not an ideal system for complex statements.

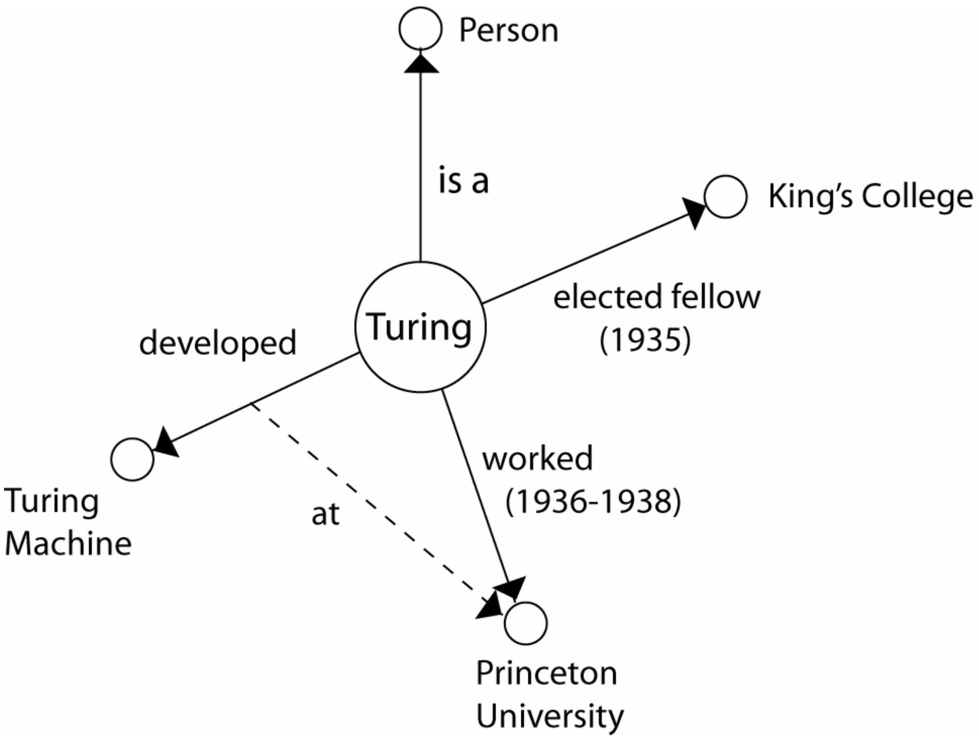


Figure 5.1. Semantic network for a set of statements about Turing.

A semantic network is another potential structure, but provides only singular relationships between entities. In Figure 5.1, we can state that "Turing developed the Turing machine", but not that it was developed *at* Princeton

University. Notice the qualification should properly be associated with the verb "developed", and cannot be introduced as a direct link from Turing Machine to Princeton University (what would this mean?).

However, rearranging these statements by placing the subject of the sentence first, we can form groups of sentences which have the same subject. Those sentences dealing with Turing as subject are grouped together:

> Turing is a person.
> Turing attended Cambridge University.
> Turing was elected fellow at King's College in 1935.
> Turing worked at Princeton University from 1936 to 1938.
> Turing developed the Turing Machine.
> Turing worked on the Turing Machine while at Princeton University.

If we represent Turing as a node in a semantic network, then each sentence with Turing as the subject is defined by the edges leaving that node (Figure 5.2).

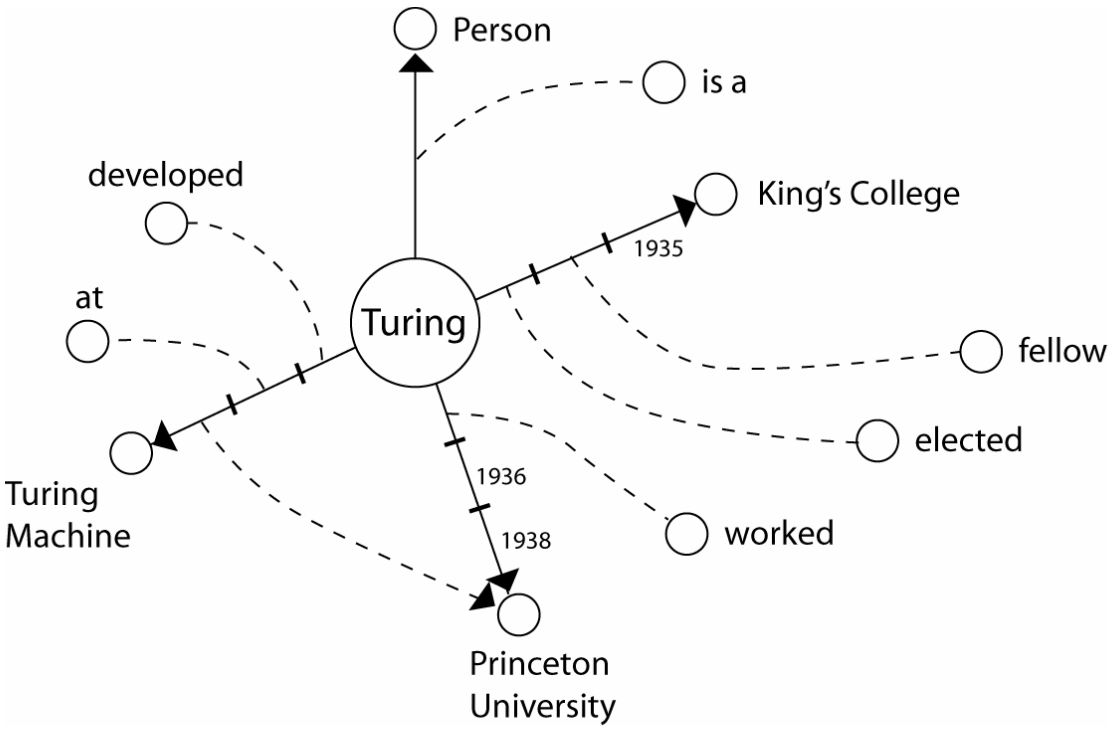


Figure 5.2. Hypergraph network for statements about Turing.

A standard semantic network does not allow us to convey the grammar of a complex sentence. This is resolved by constructing a *semantic hypergraph*, where each edge is a word-vector set with *n* words rather than a single relationship. In this way, the words contained in the edge can *also* be nodes in the graph as shown in Figure 5.2. The network is a partially-acyclic directed *hypergraph.*[1] In linguistic terms, the vectored edges allow for a richer grammar than the subject-verb-object relationships of traditional semantic networks.

[1] It is directed because the meaning of each sentence is *directional* from subject to objects and is *partially acyclic* because each edge is a vector of words while some, but not all, of these words may refer back to the first node in the vector.

The *hypergraph* is a relatively new structure for expressing semantic relationships. In 1989, Tompa first used hypergraphs to create a model for hypertext links on the Internet [5-3]. Later, Levene introduces the hypernode database to allow for semantic extensions to the standard semantic network:

> "a hypernode is a digraph structure with two built-in link types. The first link type is the arc representing a referential relationship and the second link type is the encapsulated label representing a part-of relationship. Furthermore, attributes allow us to give additional semantics to nodes." [5-4]

Notice that the hypernode database of Levene essentially represents *two semantic links,* the referential relationship and the part-of relationship. While this definition of a hypergraph allows for the addition of arbitrary information, as permitted in Levene's definition, the particular use of this structure is unspecified beyond the first two links. The structure thus operates like an extended semantic network without any organizing principle. Another use of hypergraphs is to allow for richer query structures than those provided by SQL on relational databases [5-5].

The essential feature introduced by the semantic hypergraph presented here is the distinction between the *data-nature* of knowledge as a collection of many different things and the *language-nature* of knowledge as a rich grammatic structure. By rearranging sentences to place the subject first, the vector-edges of the hypergraph can be used to expressing statements, while the overall graph structure allows us to think of objects as we would in a traditional database. Thus the hypergraph, combined with a grammatic structure, provides the ability to represent rich semantics in the format of database.

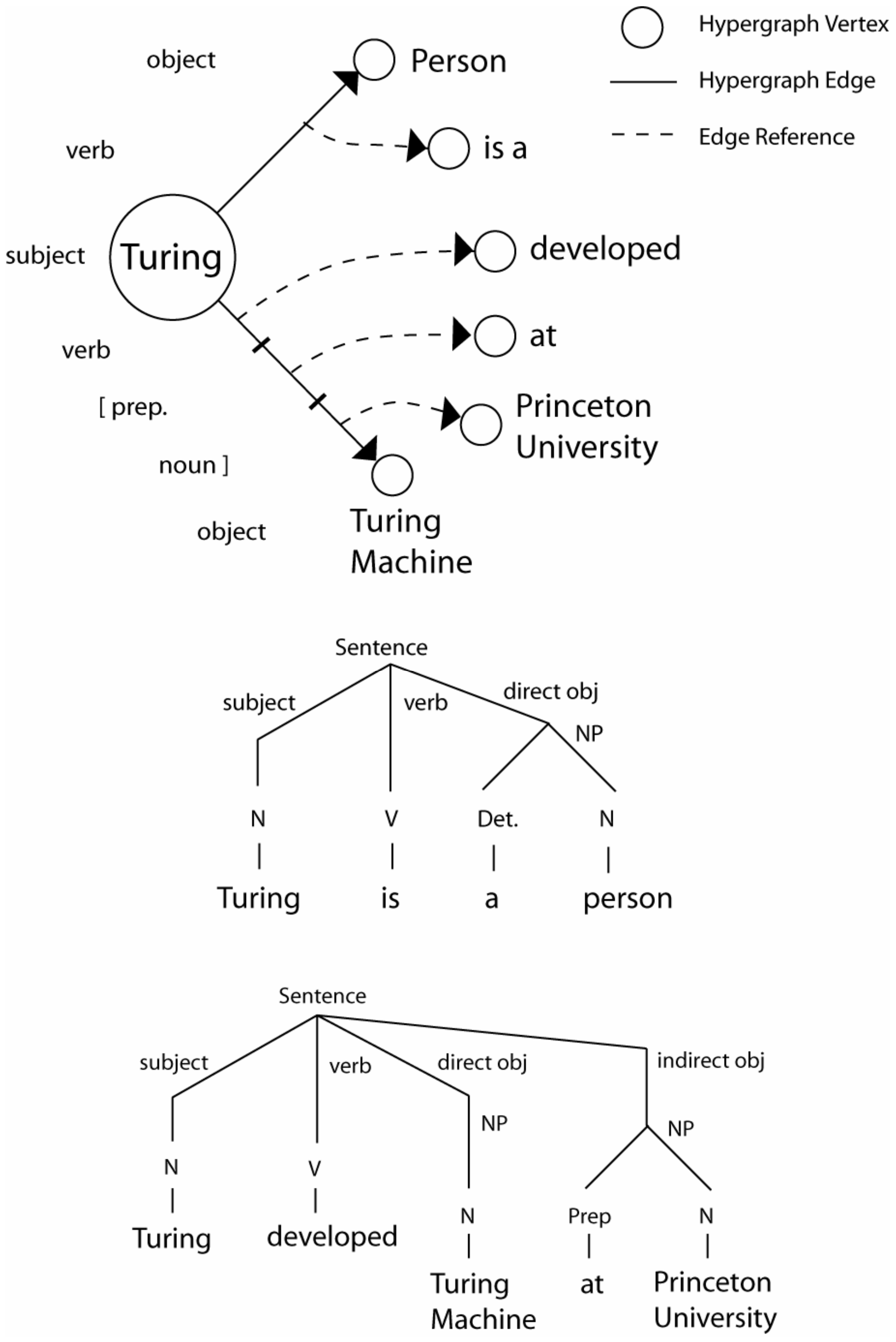


Figure 5.3. Two statements a) represented in a hypergraph with sentences embedded in the vector edges, and b) as transformational grammars

Natural language processing provides a number of structures to manipulate language. These include the conceptual graphs of Sowa [5-1], propositional grammars by Shapiro [5-6], and the generative grammars of Chomsky [5-8]. With generative grammars, noun and verb phrases are captured using cascading trees and each sentence is analyzed using a different parse tree. Using a *semantic hypergraph* we can maintain this grammar in each edge of the graph. In Figure 5.3, two sentences are represented both as generative grammars and as a single hypergraph with two edges. The type of grammar used here is a Phrase Structure Grammar [5-8]. Notice that while parse trees provide a solution to resolving word syntax, the hypergraph allows us to combine multiple statements to provide a network of relationships. The hypergraph thus operates as both a database *and* a language framework.

The language structure is contained within, subsumed, in the larger structure of the database graph. The vertices retain the quality of being *objects* of the database while the edges express *statements* about these objects. There is still an issue with word order as the parse tree of a generative grammar represent more structure than the vector-set is able to express in a single edge. A phrase structure grammar is equivalent to *regular grammar* [5-8], which is computable by a *finite state automaton* and able to produce anything that might be represented in a tree. Syntactic rules, such as parenthesis, must be introduced to flatten these structures into an edge-vector. In addition, the phrase structure grammar will be found to have certain limitations with more complex examples. Consider the statement: "Mary thought that John might not find her in their game". This example contains two second order frames ("Mary's thought" and "their game"), and one negated modal statement ("John might not"). To express more complex grammatic statements, additional syntax must be introduced. This is overcome using *layered grammar patterns* which will be present below.

As Gyssens demonstrate [5-9], each vertex in a graph database can also store an image, a sound, a date or an external file. This allows the system to maintain multimedia data in addition to written statements. In the Quanta prototype this is accomplished by having nodes that refer to image files on disk. We could also permit programming structures, i.e. C++ objects, to be the object of a sentence, thus allowing the hypergraph network to refer to *algorithms* rather than just words (Figure 5.4). By using a unique naming scheme for objects, these algorithms could be incorporated directly into the semantic network or serialized in a separate disk file.

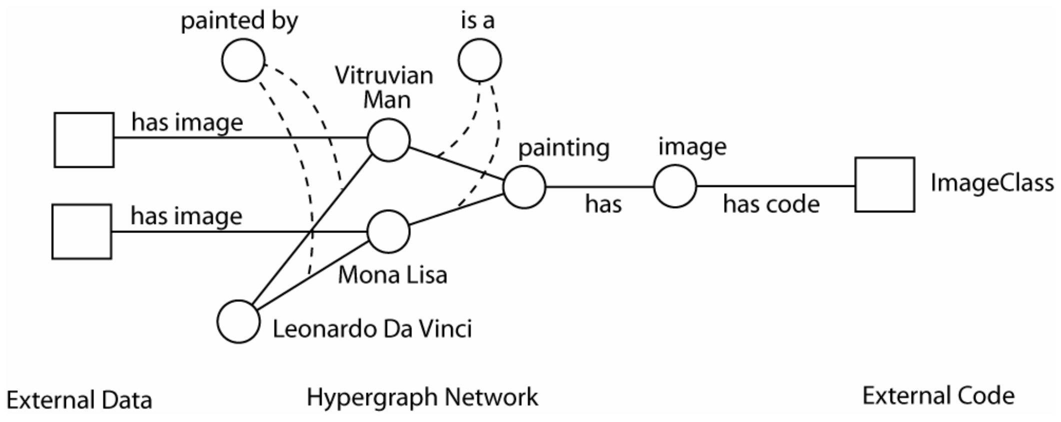


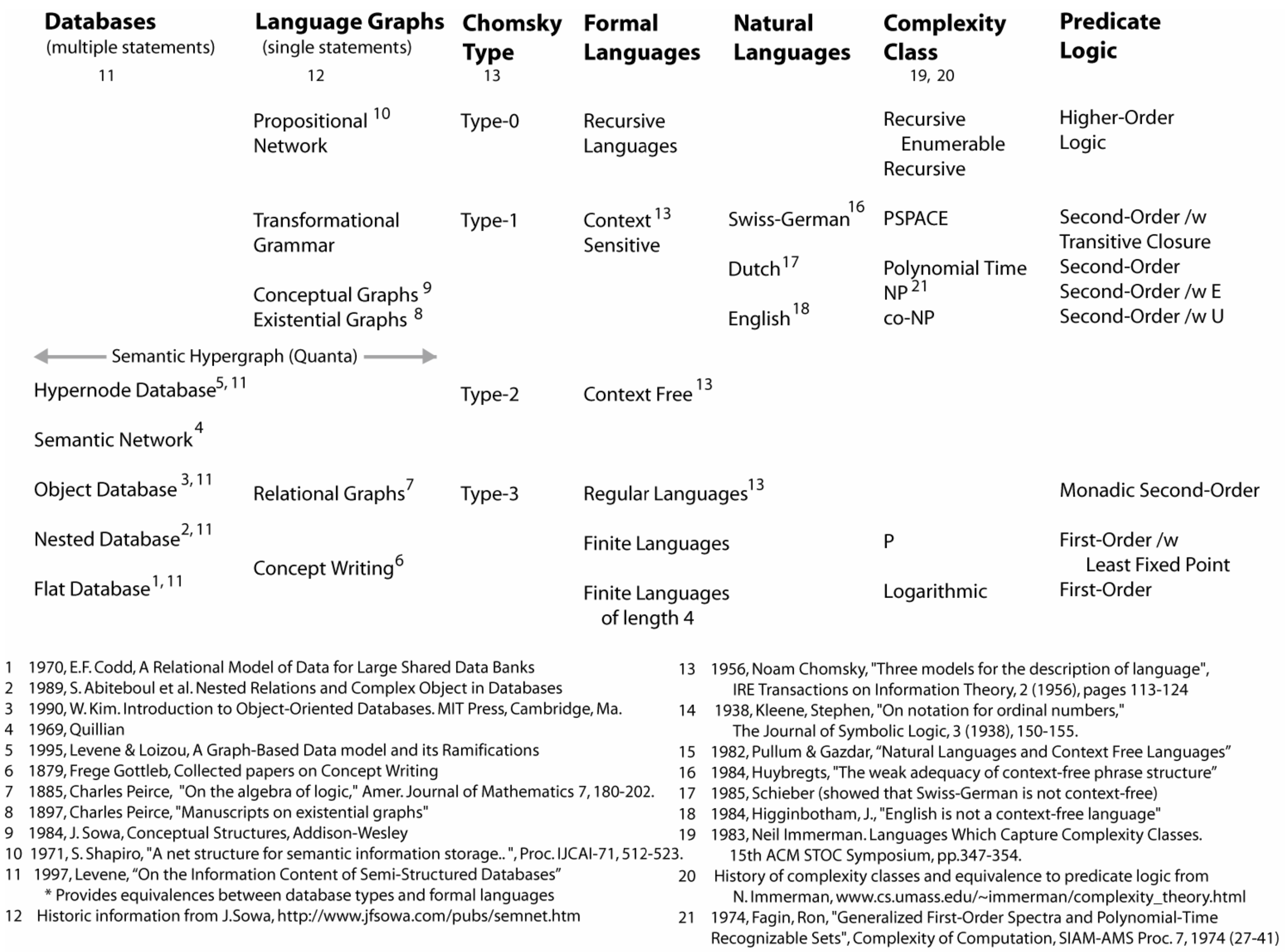
Figure 5.4. Hypergraph network with algorithms to represent a) two paintings by Leonardo Da Vinci, b) external associated digital images, and c) code to process these images.

Quanta can still be thought of conceptually as a database - of *objects* - but with complex relationships between. This database is a hypergraph of linguistic structures, but a database first since the goal is to create a general knowledge framework rather than a natural language parser.

The study of knowledge representation has a rich history as an exploration of *expressiveness*. More recently, it has been observed that the expressive power of different database systems has a direct correlation with the expressive power of various formal languages [5-10]. The exploration of complexity is an on going investigation that will not be elaborated here. However, some interesting observations can be made. First, we are just now beginning to see database systems that represent compound knowledge (multiple statements) with the same expressive complexity of natural language. Secondly, there has historically been little emphasis placed on merging graph structures for multiple statements, as used in knowledge representation, with graph structures for single statements, as used in AI. We can expect this will change.

Each of these database systems offers increased power and flexibility over less expressive ones, yet there is still a great deal of flexibility in the design of a system *at the same level of formal complexity.* The number of possible designs for hypergraph-based knowledge systems is large. Thus, we must remember that while all cars have a similar complexity their specific *design* is what determines individual performance and usability. A map of the expressiveness of databases and formal languages in shown in Figure 5.5.

It should be noted that the use of phrase structure grammars places certain restrictions on flexibility. Quanta is not capable of the expressive complexity of existential or conceptual graphs in that it cannot directly express universal and existential operators. Quanta does not presently implement full first order logic. However, its strength is the conceptual connection it forms between databases and language. With these foundations, it should be possible in the future to extend the hypergraph to implement first and second order logic.



Databases (multiple statements) 11	Language Graphs (single statements) 12	Chomsky Type 13	Formal Languages	Natural Languages	Complexity Class 19, 20	Predicate Logic
	Propositional Network[10]	Type-0	Recursive Languages		Recursive Enumerable Recursive	Higher-Order Logic
	Transformational Grammar	Type-1	Context[13] Sensitive	Swiss-German[16]	PSPACE	Second-Order /w Transitive Closure
				Dutch[17]	Polynomial Time	Second-Order
	Conceptual Graphs[9]				NP[21]	Second-Order /w E
	Existential Graphs[8]			English[18]	co-NP	Second-Order /w U
← Semantic Hypergraph (Quanta) →						
Hypernode Database[5, 11]		Type-2	Context Free[13]			
Semantic Network[4]						
Object Database[3, 11]	Relational Graphs[7]	Type-3	Regular Languages[13]			Monadic Second-Order
Nested Database[2, 11]			Finite Languages		P	First-Order /w Least Fixed Point
Flat Database[1, 11]	Concept Writing[6]		Finite Languages of length 4		Logarithmic	First-Order

1 1970, E.F. Codd, A Relational Model of Data for Large Shared Data Banks
2 1989, S. Abiteboul et al. Nested Relations and Complex Object in Databases
3 1990, W. Kim. Introduction to Object-Oriented Databases. MIT Press, Cambridge, Ma.
4 1969, Quillian
5 1995, Levene & Loizou, A Graph-Based Data model and its Ramifications
6 1879, Frege Gottleb, Collected papers on Concept Writing
7 1885, Charles Peirce, "On the algebra of logic," Amer. Journal of Mathematics 7, 180-202.
8 1897, Charles Peirce, "Manuscripts on existential graphs"
9 1984, J. Sowa, Conceptual Structures, Addison-Wesley
10 1971, S. Shapiro, "A net structure for semantic information storage.. ", Proc. IJCAI-71, 512-523.
11 1997, Levene, "On the Information Content of Semi-Structured Databases"
* Provides equivalences between database types and formal languages
12 Historic information from J.Sowa, http://www.jfsowa.com/pubs/semnet.htm
13 1956, Noam Chomsky, "Three models for the description of language", IRE Transactions on Information Theory, 2 (1956), pages 113-124
14 1938, Kleene, Stephen, "On notation for ordinal numbers," The Journal of Symbolic Logic, 3 (1938), 150-155.
15 1982, Pullum & Gazdar, "Natural Languages and Context Free Languages"
16 1984, Huybregts, "The weak adequacy of context-free phrase structure"
17 1985, Schieber (showed that Swiss-German is not context-free)
18 1984, Higginbotham, J., "English is not a context-free language"
19 1983, Neil Immerman. Languages Which Capture Complexity Classes. 15th ACM STOC Symposium, pp.347-354.
20 History of complexity classes and equivalence to predicate logic from N. Immerman, www.cs.umass.edu/~immerman/complexity_theory.html
21 1974, Fagin, Ron, "Generalized First-Order Spectra and Polynomial-Time Recognizable Sets", Complexity of Computation, SIAM-AMS Proc. 7, 1974 (27-41)

Figure 5.5. Overview of complexity in databases and formal languages.

5.2. Emergent Objects

Formally, an *object* in Quanta is a defined as follows:

Object	*Def.* A node combined with the outgoing edge-sentences of that node in which the node appears as the subject of the edge-sentence.

We can see that this definition holds for Figure 5.2 above. Every edge-sentence leaving the Turing node has the subject Turing as its first word. Informally, we can say the "Turing" concept is defined by the direct relationships between it and knowledge statements that connect it to other concepts.

The insertion of a sentence into the hypergraph requires a reorganization so that the structure of the graph is maintained. For performance reasons, it is beneficial if the subject node always appear first in the sentence. That way we can easily identify which edge-sentences apply to a particular object. However, in many cases the subject appears in the middle of the sentence. Consider the following statement and its generative grammar tree, Figure 5.6.

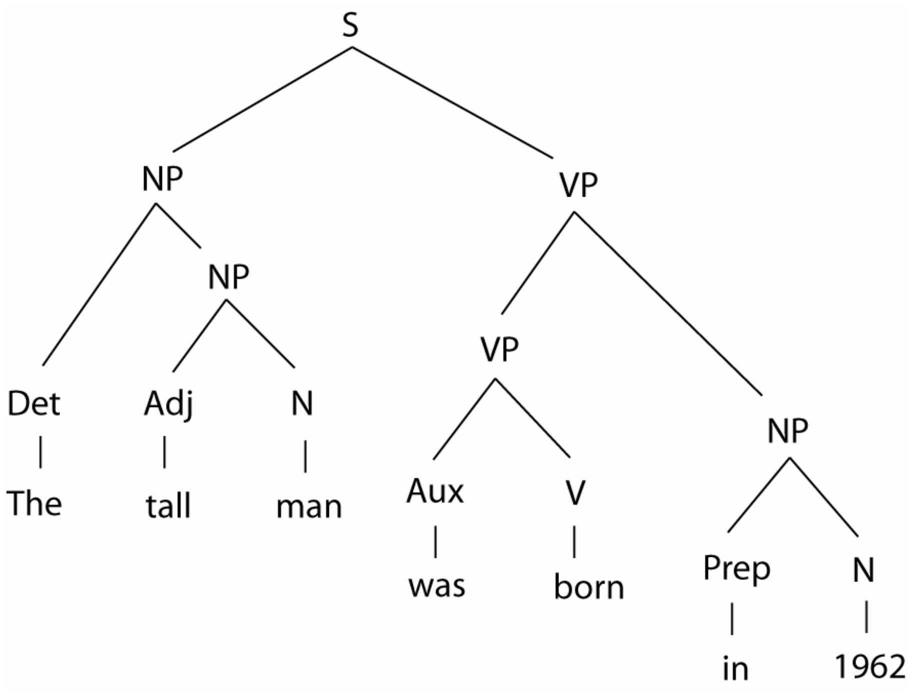


Figure 5.6. Generative grammar used to parse the sentence:
"The tall man was born in 1962".

Here, the subject is "man" yet it is enclosed in the noun phrase "the tall man". We can simplify this situation, however, by observing that unspecified objects typically never appear in a database: "The man was born in 1962" could be any man. Specifically, we wish to know *which* man:

The tall man named John was born in 1962

Placing the subject first allows the sentences to be more easily grouped for performance reasons. This is done by simply rearranging the sentence:

John is a tall man who was born in 1962.

Finally, to take full advantage of the system we would want to represent "birth", "tall" and "men" as separate concepts by transforming complex statements into several simpler ones as we have done earlier. We also add the existential relationship to define precisely what John is (a person).

> John is a person.
>
> John is tall.
>
> John is male.
>
> John was born in 1962.

This may at first seem redundant since the word John is repeated four times. However, this will be addressed with compression in how the data is actually stored. The first sentence is an existential statement of what John is. Being tall and male are additional properties associated with John, while his birth in 1962 is a event in John's life.

After constructing a moderately large set of statements in this way we can observe this restructuring of language causes certain patterns to emerge. For example, after simplification, there will be a sentence with the exact form "X is a person" for every person in the database. These patterns will be examined more careful in the next section.

5.3. Patterns

A pattern, in general, is the observation of structure in randomness. In this context it is the observation of patterns of repetitive relationships in databases. There are many kinds of linguistic patterns. Three examples are "a person is an organism", "a river is a form of water" and "Spot is a cat". The first is a pattern of *identity,* the second a pattern of *form*, and the third an *instance.* Much like programming patterns are used to construct software [5-11], linguistic patterns can be used to build knowledge relationships. Let us consider a database constructed from the following sentences:

> Beethoven was a German composer born in Bonn. He wrote his Fifth Symphony in 1804 after moving to Vienna, Austria. Bach was also a German composer born in Eisenach and wrote the Brandenburg concertos. Mozart, an Austrian composer, was born in Salzburg and wrote his Fifth Symphony in 1765.

Rearranging statements according to the rules in the previous section, we can observe the *identity*-pattern at work in this network in Figure 5.7. Nodes that represent *people* have certain common relationships with other things. People have a place and date of birth. People produce creative works. People exist at a particular location. This *object pattern* is very similar to a *class* in programming or a *schema* in a relational database. There are some important differences, however.

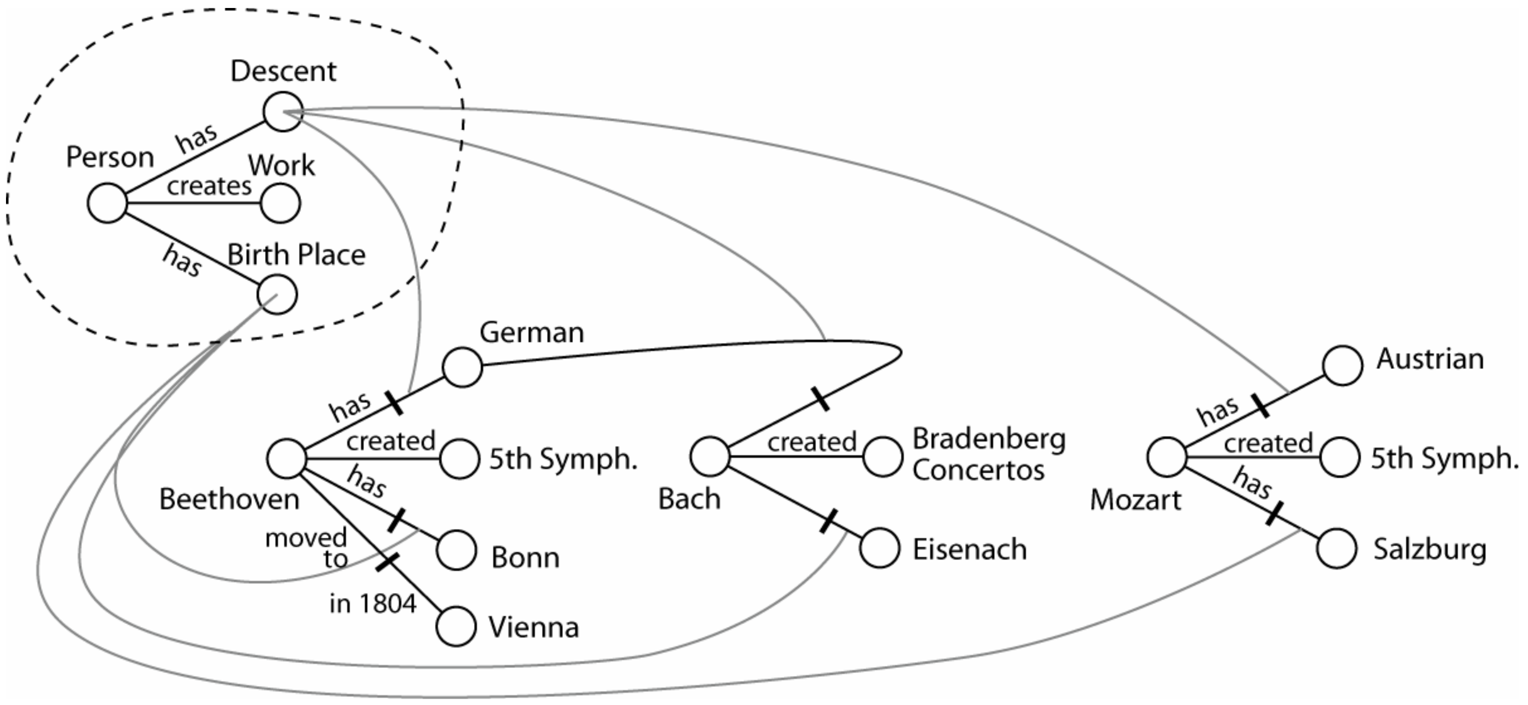


Figure 5.7. Semantic hypergraph for statements about three different composers. The sub-graph that represents a pattern for *people* is outlined.

With relational database schema and programming classes it is necessary to instantiate *every* member of the class. In addition, indices and pointers are needed to refer to the same concept. Notice that the node for German descent is only created once, so the system knows that Beethoven and Bach have the same descent. Referencing is built into Quanta based on name. Also notice that Beethoven breaks the person-pattern with the addition of information regarding his relocation to Vienna in 1804. Patterns, unlike schema, are thus *flexible* templates for generating a new set of nodes.

It is also possible to add new people to the database without using a pattern at all. The minimum required is a sentence of the form: X is a person. Finally, if one wishes to extend the concept of person to include "occupation" in Figure 5.7, it is *not* necessary to add this to existing instances of person. The hypergraph allows for a much greater degree of flexibility than relational and object-oriented databases.

In a relational database a schema is separate from its records and is normally constructed before filling the database. However, due to the grammatic flexibility of the hypergraph, we can store the pattern in the database *along with the data itself* while also distinguishing it from that data (see Figure 5.7). This allows us to create new patterns and modify them for any object without refactoring the database.

5.3. Grammatic Layers

Where there are patterns in the relationship between objects there are also patterns in the sentences themselves. Sentences that denote *existence*, such as "Beethoven is a person" and "Hydrogen is an element", have the form:

X is a Y

Sentences that denote a property of an abstract object, such as "A person has a place of birth" and "An element has an atomic weight", have the form:

X has Y

There are also sentences that convey actions such as "Beethoven composed Symphony No. 5" and "Bohr developed a Theory of Atomic Structure." These have the form:

X verb Y

There are in fact many different ways to categorize the types of sentences. Some sentences have many sub-phrases such as "Einstein developed the General Theory of Relativity from 1911 to 1915", which cannot be broken into simpler sentences since each phrase qualifies the development of the theory of relativity. This sentence follows the pattern:

X verb Y preposition Z1 preposition Z2 ... preposition Z3

The existential sentences of the form "X is a Y" are likely to be queried much more frequently than specific statements. For a biological database, the phrase "X is a mammal" is likely to be visited a great number of times as the system determines the root class of various animals. Yet the phrase "The Baleen whale has a sieve-like upper jaw" is likely to be examined only when we are investigating that particular animal.

One of the drawbacks of many current natural language parsers is that processing of simple sentences relies on the same algorithms that are used to process complex ones and therefore takes equivalent time despite their differences. In practice, it would be better to apply faster algorithms when the grammar is known ahead of time to be simpler.

To benefit from this, Quanta uses a *layered grammar* in which simple sentences are given special status by the system. Physically, they are stored the same way as any other sentence - as a vector of words - but an extra byte of information is cached with each sentence to indicate its grammatic type. This allows a parser to operate very efficiently as it can quickly determine the type of sentence and then apply the appropriate grammatic parsing algorithm to the phrase. The grammar forms that are treated in this way are shown in Table 5.7.

Table 5.2. Quanta: Grammatic layers.

Level	Purpose	Pattern	Constraints
1	Existential	N V N	where V = "is a"
2	Qualified Existential	N V N N	where V = "is a"
3	Properties	N V N N ..	where V = "has"
4	Basic Statements	N V N P N P ..	arbitrary V
5	General parsing	N V CFG	CFG = context-free grammar

N = Noun, V = Verb, P = Preposition, CFG = Context-free Grammar

Level one is a finite grammar and is the simplest to parse. In fact only three operations are needed to resolve the subject and object at this level: one to determine the grammar level, one to get the subject from position 1 and one to get the object from position 3. On the opposite end of the spectrum, level five allows for future expansion by providing space for general context-free grammar parsing. This would allow unrestricted statements in natural language that would be parsed using statistical methods. Additional syntax may be introduced in level five, using delimiters (such as parentheses), to provide more expressibility.

An important issue in parsing is the problem of disambiguation. This is solved by introducing a syntax for words with multiple meanings. A square bracket with a number is used to indicate the same word has multiple meanings. These are each stored as different nodes in the database:

bat [1]	A flying nocturnal animal similar to a rodent
bat [2]	A long, crafted stick used in the game baseball
bat [3]	The act of hitting a ball with a bat [2]

Pragmatically, ambiguity resolution simply involves selecting a node from above. A search for the word "bat" retrieves the definitions above, from which the user can select a specific meaning. As a general principle in Quanta, *all* references must be disambiguated. Thus, disambiguation of natural language should occur before data is entered into Quanta. This allows the system to function as a database rather than as natural language processing system, which may be introduced as an extension.

In practice, disambiguation can be difficult since the user may not know that other definitions of a word exist. Resolution in this case requires careful design of the user interface. When entering data, each entry of a word should also produce a search to determine which meaning is intended. Patterns can be used to help the user select the appropriate definition.

5.5. Logical Inference

The structure of Quanta has been defined as a semantic hypergraph. But what are its operations? To enter this discussion, let us consider the following query:

What constitutes the field of physics?

Several assumptions must be made. Of course, the answer is given relative to a database that has only limited knowledge of the field. Secondly, the word "constitutes" in this context is taken to mean anything that is either directly or remotely linked to the concept "physics". Finally, let us assume that disambiguation has already occurred so that "physics" represents the node for a subject area (def. #1) and not the observed motion of a body (def. #2).

Consider a sample database that contains the following sentences:

S(1) Einstein | developed | Theory of Relativity
S(2) Heisenberg | developed | Uncertainty Principle
S(3) Bohr | developed | Theory of the Atom
S(4) Copernicus | developed | Sun-Centric Model of Solar System
S(5) Newton | influenced | Einstein
S(6) Bohr | contributed to | Physics
S(7) Uncertainty Principle | is a | theory in | Physics
S(8) Theory of Atom | is a | theory in | Physics
S(9) Theory of Relativity | is a | theory in | Physics

The vertices of the database are the words of these statements. The sentences its edges.

Informally, we would expect our question to be answered on the basis of relevance. The sentences directly connected to the term "physics" constitute its immediate members. Thus, sentences S(6), S(7), S(8) and S(9) give us: the first set of terms Bohr, Uncertainty Principle, Theory of Atom and Theory of Relativity since these statements contain the word physics. We can also consider the secondary connections whose distance is two sentences from the concept "physics". Sentences S(1), S(2), S(3) give us Einstein, Heisenberg and Bohr as they contain words from the first set. Finally, a tertiary relationship is sentence S(5), which links Newton to Einstein, Einstein to the Theory of Relativity, and the Theory of Relativity to Physics. Additional orders of distance can be defined in a similar way. This form of induction has a nice property that the scope of meaning can be incrementally expanded relative to the query. As with life, there is no real boundary to the question, and this form of inductions allows use to answer it incrementally.

In artificial intelligence, this process is called the *denotive operation,* a form of procedural semantics that acts as a query on a graph [5-1]. While Sowa developed this for a conceptual graph, the denotive operator for a hypergraph is defined here using induction.

Def. *Primary relations:* The primary relations, R1(C) of a hypergraph H for a concept C are the set of unique nodes N such that there exists a sentence S in which both N and C exists in S.

R1(C) = { For all N, Exists S, s.t. N exists in S, and C exist in S }

Def. *N-ary relations*: The n-distance relations, Rn(C) of a hypergraph H for a concept C are the set of unique nodes N such that there exists a sentence S in which N exists in S and there exists a second node M such that M exists in S and in Rn-1(C).

Rn(C) = { For all N, Exists S, s.t. N exists in S, and
Exists M, s.t. M exists in S and Rn-1(C) }

This is the hypergraph equivalent to a sub-graph whose nodes are all a given path-length from the concept node C. Only the *denotive operation* will be defined here while other operations performed on conceptual graphs can be similarly constructed for semantic hypergraphs.

How do we evaluate this denotive operator for a given concept C and a given depth n? At depth one, a naive algorithm would find R1 by scanning all sentences S to look for any matches to concept C just as a person scans the above table to locate sentences in which the word "physics" occurs. At depth two or more, the algorithm would examine all sentences to see if any concept M in the sentence S is also found in Rn-1. If the database has size T, this naive implementation will have a running time for Rn equivalent to O(T^n). This means a database of one million nodes (10^6) would require 10^{18} operators just to find the nodes at level R3.

Is there a way to improve this? To find all the books in a library related to physics we need not examine every one. Instead we can use a catalog. An index for the word "physics" can be built as the sentences are added, resulting in the following:

Physics	S(7), S(8), S(9), S(10)

Indexing allows us to find related concepts much more quickly. The index lists edges directly connected to the term. Thus we can find R1(Physics) in time O(C), where C is the number of sentences connected to the concept.

Of course, we must still scan all sentences to determine which concepts are in R2(Physics) because these sentences do not actually contain the word "physics". However, *if we create an index for these nodes as well, then all search-type queries can be reduced to a localized search over the sentences directly connected to each node.*

The general solution we require is full indexing of every node. This would allow any direct query to be constant time and any higher order queries to be dependent only on sentences localized to the query concepts (rather than on all sentences in the database). The idea of full indexing could also be used in rule-based systems and to improve performance of other types of logical inference.

Full indexing is the foundation of the modern search engine for the world wide web [5-12]. All documents are scanned to create a large database of keywords and links. Every keyword then has an index of the documents it is found in. The primary difference here is that, unlike internet search engines, the keyword do not index web pages but individual sentences. Therefore, from the perspective of language, the granularity in the index is more detailed than that of a web-based search engine.

One challenge with full indexing is that it requires significant storage space. However, a semantic hypergraph is perfectly suited to a compression scheme that will allow this: dictionary compression. We store each vertex literal uncompressed [2] along with a list of *references* to the sentences in which it is found. For each sentence we store the words as a list of references to vertices. These references take significantly less space than the actual words, which are now stored only once in the entire system. In addition, the references not only aid in compression but also provide a link to other concepts and any information *they* may provide. Table 5.3 shows how to model the previous example using full indexing with dictionary compression.

[2] For words we simply store the actual word, for images and sounds we store the external filename of the multimedia content which may be compressed, and for programmatic objects we store a serialization of the object with any pointers converted to hypergraph references.

Table 5.3. Hypergraph model of nine sentences with full indexing and dictionary compression

Node	Literal	Index
N1	is a	S7, S8, S9
N2	developed	S1, S2, S3, S4
N3	theory in	S7, S8, S9
N4	Influenced	S5
N5	contributed to	S6
N6	Einstein	S1, S5
N7	Heisenberg	S2
N8	Bohr	S3, S6
N9	Copernicus	S4
N10	Newton	S5
N11	Theory of Relativity	S1, S9
N12	Uncertainty Principle	S2, S7
N13	Theory of the Atom	S3, S8
N14	Sun-Centric Model of the Solar System	S4
N15	Physics	S6, S7, S8, S9

Sent	Word-Vector	Interpretation (Not stored)
S1	N6, N2, N11	Einstein developed Theory of Relativity
S2	N7, N2, N12	Heisenberg developed Uncertainty Principle
S3	N8, N2, N13	Bohr developed Theory of the Atom
S4	N9, N2, N14	Copernicus developed Sun-Centric Model of the Solar System
S5	N10, N4, N6	Newton influenced Einstein
S6	N8, N5, N15	Bohr contributed to Physics
S7	N12, N1, N3, N15	Uncertainty Principle is a theory in Physics
S8	N13, N1, N3, N15	Theory of Atom is a theory in Physics
S9	N11, N1, N3, N15	Theory of Relativity is a theory in Physics

While full indexing allows us to easily locate the sentences associated with a node, it does not allow us to locate a node quickly given its name. Consider a search in the above data for the concept "physics". If we can locate node N15 we can easily examine its index list to identify the relevant sentences, but how do we find N15 given its name? Keep in mind the nodes N need not be in alphabetical order. The solution is to use a height-balanced search tree, which provides O (log(n)) time for both insertion and searching of nodes.

By combining full indexing, dictionary compression and a height-balanced search trees to build semantic hypergraphs it is possible to construct efficient and compact representations of complex knowledge. Full indexing allows us to perform semantic operations efficiently, dictionary compression lets us implement full indexing with little overhead in storage, and search trees should allow us to locate nodes efficiently given their names.

5.6. Storage & Performance

While semantic operations have been examined in the previous section, there are several storage issues that have not yet been addressed:

- As new nodes are added, the entire database must expand
- As new sentences are added, node index lists must also expand
- Node data itself must be permitted to change or expand

Ideally, we would like all of the features of the hypergraph to be flexible. New and changing data should not burden the system. Practically we must store two lists: 1) The list of vertices and their indices and 2) The list of sentences.

In addition, we must also maintain a height-balanced tree for alphabetic indexing. Each of these lists must be permitted to change as needed.

To solve these problems, Quanta implements a hierarchical file system with heap-based node allocation. The hierarchical file system separates the database structure into vertices and edges while the heap-based allocation allows individual nodes and sentences to be reallocated as needed. The file system allows for three different types of files: 1) Data files, 2) Node files, and 3) a Meta file. The meta file stores allocation information about the other files. It is essentially a directory of the other files (including itself). Data files contain raw data without labeling of data blocks while node files keep track of individual pieces of data.

These are not literal files at the operating system level. They are referred to as files, however, because they serve a similar function. Rather the custom software of Quanta stores all of these "files" in a single physical disk file on the hard drive of the native operating system. One reason for this is the expectation that the database may become quite large. The prototype database contains roughly 8000 vertices and 12,000 sentences. While not significantly large, no limit has been reached yet. In the future, it should be possible in theory to split the hypergraph into multiple files or even distribute it among machines provided the vertex indices are properly maintained.

Only the height-balanced tree is kept in physical memory. This was done for performance reasons and also ease of implementation. Keeping the search tree in memory is not ideal as this will present limits when the database grows. Future designs would incorporate the height-balanced search into the data file on disk so that memory can be utilized for temporary visualization structures.

Table 5.4. summarize the various data storage layers and algorithms used in the Quanta prototype.

Table 5.4. Quanta: Data storage layers and algorithms

Layer	**Problem**	**Solution**
Semantic Network	Knowledge database	Hypergraph network
Graph Storage	Query performance	Full Indexing
	Full index storage	Dictionary compression
Node Storage	Node search and retrieval	Height-Balanced Tree
	Node expansion and addition	Heap-Based Allocation
File Storage	Vertex and Edge Storage	Hierarchical File System

The investigation into mathematical definitions of logical operations on the hypergraph, and the performance issues on network inference is superficial here. Other inference operations include *modus ponens*, *universal generalization*, *universal induction*, and *deductive inference*. While the performance of these algorithms is not investigated, the use of full indexing should eliminate any global searches of the graph. Just as various amounts of effort are needed for people to answer questions related to more abstract concepts, the nodes traversed will be limited to subsets defined by a specific query. Higher degree nodes will require more processing than lower degree ones, but no cases should all nodes be traversed.

The Quanta system was developed as a prototype to test the feasibility of hypergraphs to store grammatic structures. This is used, in the following chapters, to build an interdisciplinary ontology and a set of novel visualizations of semantic data. A more thorough investigation of a long-term strategy for Quanta would involve a distributed prototype incorporating some of the additional features mentioned above (e.g. disk-based search trees), and theoretical and empirical measurements of scalability and performance. More details on the implementation of Quanta are described in chapter nine.

5.7. Representational Layers

Logically speaking, language may be thought of as a set of ideas that build from words to sentences, sentences to objects, objects to patterns, patterns to instances, and instances to classs. This sequence defines the *representational layers* of Quanta (Figure 5.8).

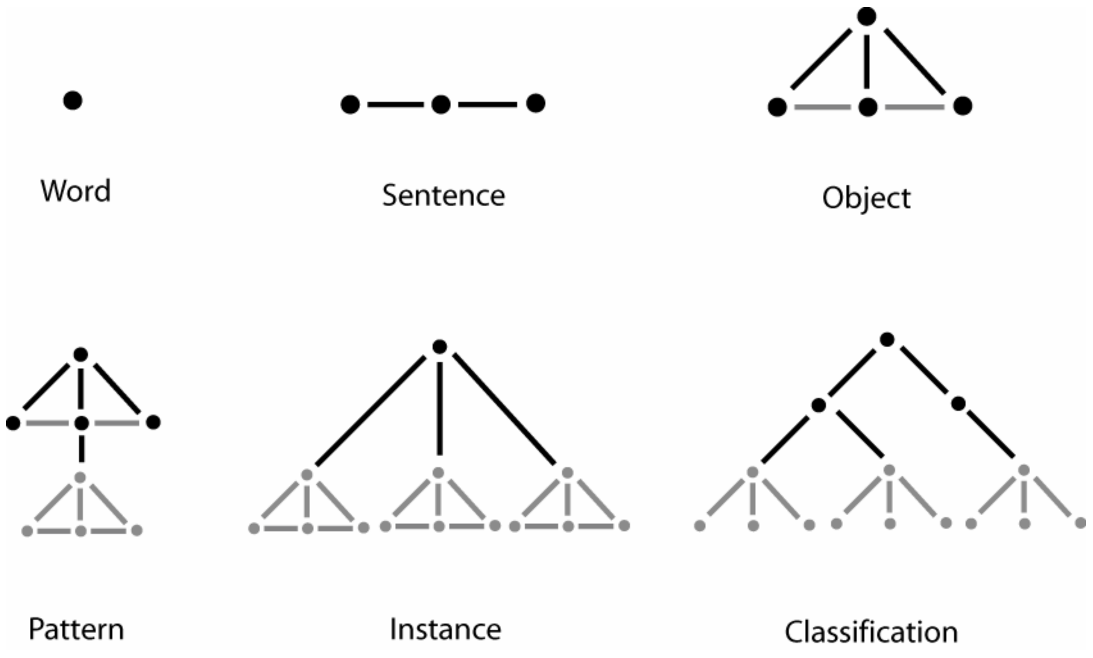


Figure 5.8. Patterns of knowledge representation at different scales. The layers demonstrate increasing complexity in knowledge representation.

As with any system there are limitations. Current limitations of Quanta have to do with the expressive power of sentences which are restricted to phrase structure grammars. A design-driven approach to this is taken by construct a formalized English in *grammatical layers,* so that simple grammars can be operated on efficiently while more complex grammars can be developed in the future as needed.

One key benefits of the semantic hypergraph is that any object can be created, extended, classified and elaborated indefinitely. Patterns can be used to create databases of objects, while the patterns themselves can be created and expanded for any concept. As with semi-structured databases, there is no formal distinction between schema and data. Classifications can be built into the network as well. Objects can be instances of a class, or belong to multiple classifications simultaneously as we will see in the next chapter (Ontology & Classification). The system can thus be used to represent many different kinds of knowledge across multiple disciplines.

Similarly, *layered grammars* allow the concept of grammatic complexity to be efficiently expanded as needed. Basic relationships, such as identity, are processed with fast, fixed alogrithms. More complex grammars can be embedded in the graph edges as needed. Thus, the grammatic power of hypergraph databases is open to future development. Indexed, dictionary compressed hypergraphs operate on knowledge at a lower level than current textual approaches to metadata on the semantic web, thus affording the benefits of database performance and stability as well as semantic flexibility.