

A Term Logic for Physically Realizable Models of Information

Steven Lindell

Department of Computer Science
Haverford College, Haverford PA 19041
slindell@haverford.edu

Abstract

Fred Sommer's revival of traditional logic is based fundamentally on the belief that a classical approach to deduction, based closely on natural language, is more faithful to human reasoning. Most often, this contribution is seen within the realm of philosophy and linguistics. One of the main purposes of this paper is to show that traditional logic can be applied to study the foundations of computation seen through the eyes of natural philosophy – mathematics and the sciences. We analyze first-order definability over classes of mathematical structures which represent scalable models of information, using traditional logic. By considering the physical requirements of storing arbitrarily large amounts of information, we discover that the scalable data models are those that consist of networks whose nodes are of bounded size and degree. This condition arises naturally from the material limitations on the spatial distribution of data in memory (essentially the bounded density of matter embedded in a finite dimensional space). Our main result is that on special uniform classes of finite models, first-order logic with equality is equivalent to a considerably simpler variable-free logic in classical subject-predicate form. This term logic incorporates an innovative combination of numerical quantifiers and direction functions, neither of which disturb the essential monadic quality that is characteristic of traditional logic.

0. Conceptual introduction

Traditional logic has nearly faded into obscurity in the twentieth century, eclipsed by tremendous developments in modern logic. The purpose of this volume is to recognize the pioneering work of Fred Sommers, who championed its revival on philosophical grounds, by articulating its familiar correspondence with natural language, and emphasizing a deductive system that is cognitively more similar to human thought. Indeed, a beautifully succinct summary of this view is that Sommers believes in “the classical idea of having a psychologically realistic theory of reasoning based on ordinary grammatical analysis.”[†]

Inspired by his approach, one of the primary aims of this paper is to show that traditional logic may provide important insights when describing properties of symbolic information represented in physical form. The goal of showing that classical subject-predicate term logic may have application to fundamental principles in the philosophical foundations of computation can be seen as a steppingstone toward a “physiologically realistic theory of computing based on ordinary informational analysis”. (Here, the term ‘physiological’ is being used in its more obscure but original meaning of “pertaining to the material universe or to natural science”.)

Although originally inspired by the close connection between traditional logic and natural methods of representing information, we hope to show how an enhanced symbolic version might also provide a more perspicuous tool for the analysis of first-order definability over physically scalable models of information — identified as highly

[†] from Johan van Benthem's review of [Sommers], *Philosophical Books* 24:2 (1983), pp. 99-102.

uniform classes of finite structures. In bridging an immaterial view of data with its material realization in memory, we hope to shed light on the ultimate limits that a physically constrained mathematical theory must satisfy. After all, human thought is also presumably physically based (though probably not indefinitely scalable).

Computer science

Computer science is unique among the sciences in that it encompasses both *software*, abstract ideas whose underlying theory is mathematical, and *hardware*, material objects governed by physical law. The fundamental mathematical theory behind software can be broken down further into *data structures*, which model the information stored in a computer, and *algorithms*, the computational processes that transform that information. Algorithms are always developed so that in theory they could operate on any amount of data, given sufficient time and space to do so, and efficiency ratings (computational complexity) are based on their asymptotic consumption of these resources (time and/or space). Computers are complex machines carefully engineered to execute algorithms, storing and processing information according to precisely specified rules of behavior. When provided with a program which defines those rules (the algorithm) and the input on which to operate (the data), these theoretical aspects can materialize in a hardware implementation, requiring *matter* for information and *energy* for computation. The focus in this paper will be on the informational aspects, while computations will be the subject of a subsequent paper.

When information is stored as data in memory, it takes on properties of *size* and *shape*. The amount of data that can fit is limited by the memory's capacity, and how that data is arranged limits the memory's access speed. Aside from reliability, capacity and speed are the two most important aspects of memory performance.

To acquire a deeper understanding of this, we will embark on a theoretical analysis of how information makes its way from logical structure to physical representation. What possible connection could there be between the mathematics of information and the technology of data storage? The answer can be found when the quantity of data exceeds any predetermined bound.

Section summary

The paper begins in Section 1 by arguing how the material requirement of storing arbitrarily large amounts of information at bounded density necessitates an asymptotic view of how data is shaped in space. In section 2, we illustrate the various ways that information in the form of relational facts can be represented mathematically. In section 3, we explain the various methods of memory organization that are used to store data physically. Section 4 describes the uniform symbolic structures we are using as models of information, upon which our specially designed logic will work. This logic, consisting of functionally defined terms which are numerically quantified into statements, is detailed in section 5. Immediately following is our main result in section 6 which demonstrates equivalence with first-order definability. The paper is summarized in the last section, 7.

Other than a basic understanding of first-order logic, no additional background in mathematics or physics is required to understand this paper.

1. Philosophical foundations

Plenty of room at the bottom

In 1959, Feynman gave his now famous lecture on the hitherto enormous untapped potential of nanotechnology to store information at incredibly high density, challenging engineers and physicists to reproduce the entire contents of the Encyclopedia Britannica on the head of a pin [Feynman]. He showed that there was more than enough room to accomplish what seemed at the time like an incredible task. Indeed, this much information storage is already available in a consumer device about the size of a quarter, and the ability to write atomically has already been demonstrated in the laboratory. All of this has been made possible by the exponential growth in computer technology, currently doubling storage density in under a year. What this means is that the size and cost to store a *bit*, the smallest unit or datum of information, has been periodically halving due to improvements that have made it possible to read and write data at progressively smaller scales of distance using ever smaller amounts of material.

But not an unlimited amount

This rapid growth in information density might lead one to believe that there are no practical limits to the amount of data that can be stored in various types of storage devices. For example, in under a decade, it will be possible to store the entire contents of the Library of Congress in a device the size of a postage stamp. But despite these startling accomplishments, such increases in density cannot go on forever. In just a few decades, we will reach scales at which the very structure of matter itself prohibits further progress. No amount of cleverness will be able to exceed these limits without breaking the known laws of physics. At these minuscule scales, when we have reached the density limit of matter, the ideas put forth here begin to have application.

No room to go but out

There is an obvious impact on the field of computers because of a fundamental philosophical difference between mathematics and physics: the quantum nature of our universe implies that there is a smallest scale (say the Planck length); but the infinitude of natural numbers implies there is no largest number. No arbitrary limit can be placed on the amount of data that a mathematical model of information must deal with: it is clearly boundless. But the amount of matter that can be confined to a given amount of space is not. When storing arbitrarily large amounts of information, there is nowhere to go but out.

Large numbers

As human beings we are used to dealing with numbers in ordinary life up to a certain size, and extrapolating from those experiences to assume that operations such as counting could, at least in principle, go on forever. From these simple observations we can deduce at an early age that there is no largest integer, even if the notion of infinity does not yet sit entirely comfortably with us. But in this paper we need something more; a fact about numbers that is not often appreciated by even adults. Just as there is no largest number, there are no large numbers. It is only by context that we can assign relative notions of largeness to numbers. In an absolute sense, compared with the unbounded infinitude of

all numbers, no number can be deemed mathematically large.

The importance of scale

Though simple to state, this principle can be hard to accept. Perhaps it is because most scientific endeavors study phenomenon at some scale, even though that scale may be extremely small (say particle physics) or large (say astronomy). On the other hand, mathematics does not even concern itself with scale, being as comfortable with one real number as any other. Mathematics also deals with ideas of arbitrary, even infinite, size. But the difference here is more substantial – in most situations there is no “implementation” envisioned. Even in areas of mathematics where computational techniques are available, “experiments” are necessarily finite approximations of more general cases that involve arbitrarily large discrete instances or continuous precision.

Why should the existence of physical limits place any restriction on mathematical ideas? The answer is it doesn't, until one wants to implement those ideas. Basically, this is what computer science is all about: taking abstract objects and making them real. This is different than engineering where one wants a bridge of a certain length, a rocket of a certain size, or a train that can go a certain speed. There is never the intention that the techniques should or could scale to an arbitrary degree. But this is exactly what happens in computer science. This requirement, to operate at *any scale*, illustrates the fundamental distinction between computer science and the rest of the sciences.

The role of the computer

Computers are somewhat unique in that they implement abstract mathematical concepts in the concrete physical world. It is not sufficient to have an idea on how to solve a problem – its efficacy must be demonstrated in the material world even if only on small examples. In part, this explains the importance of the "demo" which replaces the role of the experiment in the natural sciences [Hartmanis]. One can then argue by extrapolation how the implementation could be extended, at least in principle, to work on problems of any size. This will be an important guiding principle for what follows. For to really understand information and computation, we must also be concerned with the physical constraints nature places on their realization. Theoretical models should include their instantiation in the computer – in effect building in the tools required to carry them out.

Limits to asymptotic shape

The existences of limits imposed by nature are all that concern us here, not their actual values. All we really need is to acknowledge that matter cannot be compressed to arbitrarily high density. That is to say, there must be a fixed finite bound on the amount of information that can be stored in a given amount of space. This simple observation has profound implications for how data can be structured. Huge collections of bits cannot be arranged in completely arbitrary ways. Their imposition in space yields a more limited variety of possible models, which will be studied in this paper. In other words, since no amount of information is large in any absolute sense, *the physics of storing arbitrarily large amounts of information determines the mathematical form of that information*. I.e. size ultimately influences shape.

So in the next sections we proceed from the abstract mathematical structure of

information, to its concrete material realization in physical space.

2. The Mathematical representation of Information

Information can be seen as a collection of facts which impart knowledge by their context. Hence, an individual piece of information can be viewed as a *fact* over a given domain of discourse. Each fact relates one or more individuals of its domain. An example is:

- Jane is married to John

An information structure consisting of facts like these is represented mathematically by a *relational model*. In all such models, facts about a domain of discourse are represented as *tuples* between those elements. E.g. “is married to” between people can be written *Couple*, and the pair given above written symbolically as *Couple (Jane, John)*. Similar types of facts can be collected together to form a *relation* such as *Couple = {(i, j) : i is married to j}*, which are just sets of tuples whose common length determines the *arity*. Relations are categorized by their arity, or how many elements they relate (*Couple* has arity two).

The role of logic

Logic is a formal language for expressing properties and explaining conclusions about the information being modeled. When a property is defined by a question it is called a *query*. For example, we might wish to check whether *Couple* represents a matching between its members:

Query: Is each person married to exactly one other person?

In this paper we are only concerned with queries about static situations of information structures. First-order logic is well suited to this job, expressing many (but not all) properties of interest. A subsequent paper will focus on a fixed-point logic extension to cover dynamic properties in order to capture computations. Basic background on relational models from the computer science perspective can be found in any standard book on databases such as [Ullman]. Basic foundations of mathematical logic are covered in any introductory textbook such as [Enderton].

Representing relations

Relational models can be represented in various ways. In a *structure* they are abstract sets of tuples, in a *database* they are tables, and in a *graph* they are collections of nodes.

Abstract method: Finite Structures

Since we are studying stored information, we assume relations with an arbitrarily large but finite number of facts, giving us a finite domain. Given a fixed collection of relations R_1, \dots, R_m , where a_i is the arity of R_i , a *finite structure* $\mathbf{A} = \langle A, R_1, \dots, R_m \rangle$ which combines them is said to have signature a_1, \dots, a_m (its total arity a is just the maximum arity of its relations). It will be sufficient for our purposes to assume that the domain $A = |\mathbf{A}|$ is implicitly defined to be the set of elements that participate in any of the relations,

and that the number of (distinct) such elements $n = |A|$, is defined to be the *size* $\|A\|$ of A .

Of course, we need a polynomially larger number of tuples in order to take into account the actual amount of physical storage required to represent A . Most often, what we are really interested in is not the precise amount of space, but rather its asymptotic growth rate with respect to $|A|$, up to a constant of proportionality. So in actuality A would require $\text{SPACE}[n^a]$ to store. A graph (one binary relation) on n vertices could contain up to n^2 edges and hence require a quadratic amount of space. For example, in a group of $n = 6$ people (vertices), there are up to $n(n-1)/2 = 15$ ways (edges) for them to shake hands, as illustrated below.

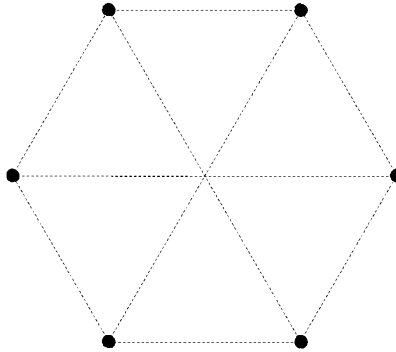


Diagram depicting how n objects can be related by a quadratic amount of information

First-Order Logic

In order to derive information from a structure, there needs to be a way to bring facts together. The standard language for describing properties of finite structures is First-Order Logic, or FOL, allowing simultaneous discourse about a variety of facts. E.g. suppose the *Couple* relation above is represented symbolically by C , so that

- $(\text{Jane}, \text{John}) \in C$

Continuing with the marriage example, a formula that expresses unique correspondence is:

$$\neg \exists z [\exists x C(x, z) \wedge \exists y C(z, y)] \wedge \neg \exists xyz [y \neq z \wedge C(x, y) \wedge C(x, z) \vee C(y, x) \wedge C(z, x)]$$

The first part says that no element appears on both sides of the relation, and the second part that no element is related to two distinct elements on the opposite side of the relation. We basically want to say that nobody is married to oneself, and that no one is married to two different people. This boils down to simply saying that everyone currently participating in marriage does so exactly once.

Tabular method: Relational Databases

The information in a finite structure is contained in its relations; specifically in the tuples of domain elements it relates. In practical applications, it is convenient to arrange this data in the form of tables, one for each relation. In a given table, there is one named column for each place in the corresponding relation. Hence, the number of tables and

their widths is fixed by the signature. Within tables, each row represents a fact; a tuple in the relation. Each entry in a row is simply a label for an element of the domain (the labeling scheme is irrelevant as long as distinct labels are assigned to distinct elements). Storing information this way is known as a relational database (there are additional details but they are superfluous). It becomes easy to see that the size of the database (measured by total number of entries) is proportional to the number of facts being stored.

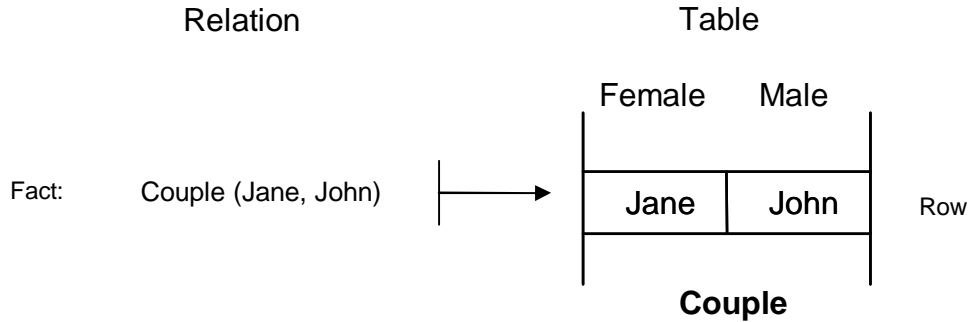


Figure: a tuple in a relation becomes a row in a table

Structured Query Language

One area of application in computer science where logic and information have combined in a particularly successful manner is the field of relational databases. It has become standard to obtain information from a database by writing a query in *Structured Query Language*, or SQL. Although it is not necessary for us to deal with SQL code in this paper, it is important to realize that queries formulated for database programming are based on relational algebra, which is itself equivalent to FOL, known as the relational calculus [Ullman]. Hence, first-order logic provides a foundation for the study of query theory – the fundamental underpinnings on how to access information stored in a database.

Graphical method: Data Structures

To more clearly illustrate how a database might be stored in the memory of a computer, we utilize a representation in which rows of a table are converted into vertices of a graph. Nodes represent tuples, and are labeled by the relation they are part of, with ordered links to the elements they are relating. Domain elements are nodes too, but without labels or links, and are called atoms. (Note that many nodes may point to the same atom.)

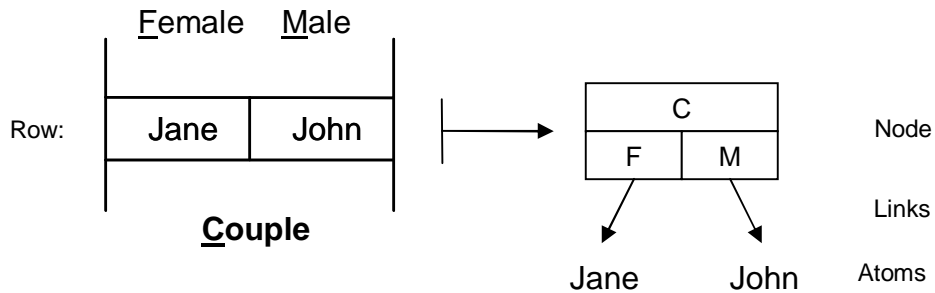


Figure: rows from tables become nodes in a data structure.

It is easy to see how facts are inferred by immediate adjacency between nodes. For example, in the figure, fact *Couple (Jane, John)* is determined from the existence of a node labeled *C* whose Female and Male links point to *Jane* and *John* respectively. The idea that knowledge of facts can only be based on the proximity of elements being related is how we are going to connect each datum with its physical embodiment.

Language: Singulary Logic

The advantage of this graphical method is that it accurately shows how to convert an arbitrarily large amount of relational information into a structure which stores all its data in a uniform fashion, each fact being tied to an object that contains only a bounded number of items inside of it. Each of these nodes contains a relation symbol and links to the elements being related. To make the internal structure of each node identical, some (or in the case of atoms, all) links might be empty so we have them point to a special atom *nil*. This yields a structure whose size (number of nodes) is precisely equal to the amount of information being stored (the number of elements being related, with *nil*, plus the number of tuples relating them), all without losing any information.

But there is yet more. It is possible to model these data structures in a particularly simple way: a signature with only unary relations and unary functions. The terminology *singulary logic* has been used to describe such a vocabulary [Church]. So now we have something whose size is at least roughly proportional to the amount of physical memory it might occupy. It is not difficult to show that every first-order query on the original database can be translated into a (typed) first-order formula (with equality) on the new data structure.

Referring to the figure in the above case, this is accomplished by making the following substitution:

$$R(x, y) \mapsto (\exists z) [C(z) \wedge F(z) = x \wedge M(z) = y]$$

Note that many variables can occur together in the same formula even though the elementary symbols are themselves singulary. Later on, we will obtain what is essentially equivalent to a mono-variable logic. This only occurs in special situations which are consequences of requirements that enforce physical scalability (basically where the in-degree of each atom is bounded). To understand how that can happen, we must turn to an analysis of how information is actually stored in a computer.

3. The physical storage of data

It is generally accepted that in mathematics, any consistent requirement on the existence of an object can be witnessed by a (potentially infinite) set. This is fine for an abstract area like mathematics. But to be faithful to reality, information must have a material representation determined by the physical properties of the media in which it resides. So although it might be more elegant to study the concept of information independent of its representation, information must ultimately have a physical form. This section is concerned with the mechanics of how data is actually stored in memory, and its theoretical consequences.

Information theory

A beautiful probabilistic theory of information was developed by Shannon over fifty years ago to understand the amount of data required to reliably store or transmit redundant information in the presence of noise (see [Hamming] for an elementary treatment). This quantitative theory does not concern itself with how data is structured; usually assuming it takes the form of a binary string (a linear ordering of bits) because of the sequential nature of time in communications. However this configuration could really be any rigid arrangement of symbols from a fixed finite alphabet which is previously agreed upon by sender and receiver (such as the rectangular grid of color RGB pixels in a television screen). The only provision would be that the specified arrangement must be asymptotically scalable to support any (finite) amount of symbols. A linear sequence intuitively satisfies this constraint.

Arbitrary amounts of data

Our goal is to determine which classes of relational models admit practical implementation asymptotically. To accurately reflect the material requirements of storing *any amount* of information, we demand there must be a uniform manner of storing a sequence of finite objects of unbounded size using matter in space, thereby taking into consideration the practical necessities of utilizing data at arbitrarily large scales.

We have already seen how, in a fixed vocabulary, relational information of arbitrary size and complexity can be represented by data structures – finite objects containing uniform collections of nodes which hold symbolic data and links to other nodes. To actually store this information requires a memory in which arbitrary arrangements of pointers can be embedded. This implementation is made possible by a powerful memory architecture in which every place in memory is immediately accessible from any other place by an access mechanism known as *addressing*.

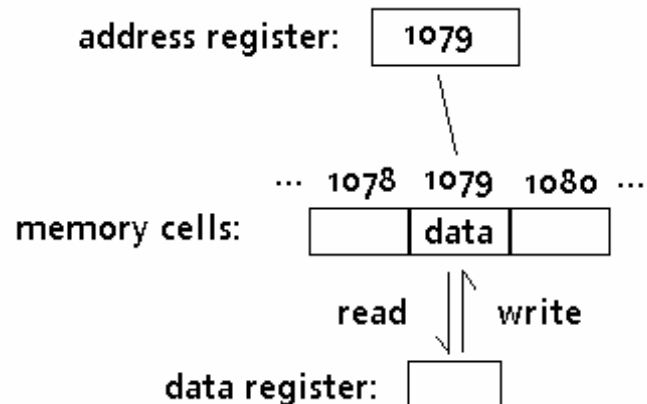


Figure depicting the logical layout of data in a pointer-based memory, showing that an address can reference any numbered location in memory. N.b. locations are arranged linearly for purposes of illustration only; but they are really located in a hyper-dimensional space since cells are numbered and accessed by binary addresses.

Although conveniently appealing, we will see that there are difficulties in implementing this idea at arbitrarily large scales.

The memory hierarchy

Alan Turing introduced the first formal model of mechanical computing in 1936, originally to study the mathematical limits of computation. This model had to have the potential to manipulate arbitrarily large amounts of information. Designed to be as simple as possible, it used a linear “tape” with a semi-infinite sequence of cells, each of which could hold one of finitely many symbols. A finite control reads from or writes to cells of the tape based on its state, modifying that state and moving left or right.

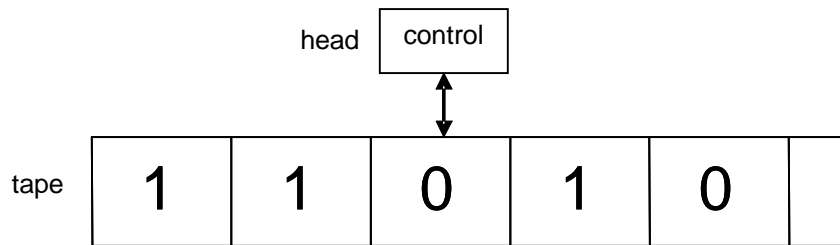
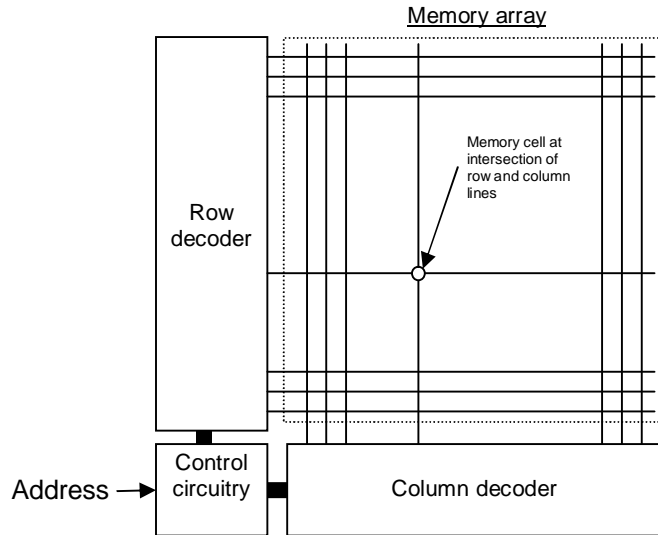


Figure of a Turing Machine

However, he quickly observed that in a computer whose memory was based on a linear sequential structure of cells, accessing the data stored in it would be unbearably slow [Turing, p.7]. So he subsequently proposes an addressing scheme using a system of switches (akin to how a telephone network routes calls), which is conceptually equivalent to the way modern digital computers access an exponential amount of data compared to the address length. However, Turing carefully notes that this method runs into problems when addresses can no longer fit into a single location, a topic which we will return to shortly.

Main memory

The current state of technology allows computers to use this addressable structure for their *main memory*, where it is called *random-access memory*, or RAM for short. It got this name because the speed at which data is accessed does not depend on its location. In actuality, as can be seen clearly in the figure below, the physical arrangement of a modern memory chip is two dimensional, with rows and columns which access the individual electronic cells.

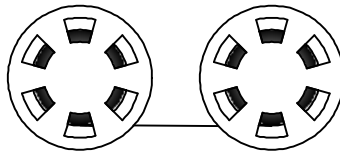


(a) block diagram of the physical layout of data in random access memory

With sufficiently high density, it is possible to make time differentials between distant locations small enough to be insignificant. However, it should be clear that in any finite-dimensional space, speed of light limitations prevent the construction of a memory which is always random-access, regardless of size. For this reason, computers need alternative methods of mass storage that rely on the inherent spatial arrangement of data.

Secondary Storage

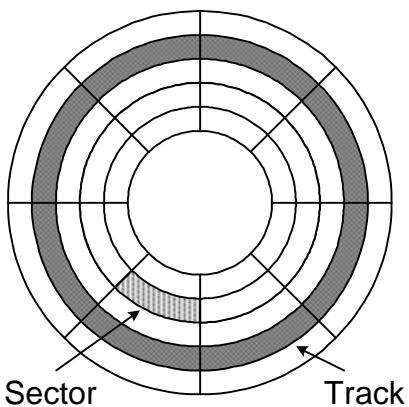
Information which cannot fit in RAM is delegated to *secondary storage*, designed primarily for capacity, not speed. Secondary storage is usually much slower due to its sequential nature of accessing data. An ordinary magnetic *tape* cassette used for audio or video is a perfect example of information that can only be accessed sequentially.



(b) linear tape (on reels)

Locations are linearly ordered in one dimension, so accessing one location from another requires scanning all intermediate locations (fast-forward and rewind). Indeed, early computers used tape storage extensively until more convenient methods utilizing a two-dimensional method became less expensive.

In the most familiar form of mass storage device in use today, an optical *disc*, radius and rotation are varied to access data stored in tracks and sectors respectively.



(c) disk storage in tracks and sectors

Because they offer much faster access times than tape, audio (CD) and video (DVD) discs are often erroneously referred to as being random-access. In reality, their asymptotic speed can only approach the square root of their capacity, which is still significantly better than a purely linear arrangement.

Future developments promise techniques that utilize three-dimensional structures such as crystals or holograms, further improving both speed and capacity. But the asymptotic performance of a true RAM (essentially logarithmic access time) cannot be achieved in a finite-dimensional space.

Performance issues

Special techniques involving caching have been developed to smooth the transition between different levels of the memory hierarchy to accommodate storage technologies that trade off speed for capacity. Since we are so very far away from the maximum storage density of data, one might think that such distinctions are a temporary artifact, and that RAM could eventually be scaled to any size, making it possible to store all information in high-speed memory. However, as we have seen through a closer investigation, there is sufficient cause to cast doubt on this assumption.

Scalability

Perhaps the first person to recognize this was Alan Turing, in the very same breath as the discussion cited above [Turing, p.8]. Even in proposing the now ubiquitous idea called RAM, Turing himself recognized the theoretical infeasibility of such an approach. Eventually, he argued, no matter how cleverly one engineered the memory of such a “practical computing machine”, a memory of arbitrarily large size must eventually degenerate into using sequential access to reach its data elements. In other words, the notion of immediate access to arbitrarily large amounts of information via an addressing scheme does not scale without breaking some law of physics. Actually, because of the exponential growth rate in technology, we are quickly (in a matter of decades) reaching those physical limits. (A relatively recent scientific investigation of the hurdles that current magnetic storage technology needs to overcome is presented in [Toigo]). Therefore, it will become necessary to take into account physical limits in our mathematical models of storage.

4. Models of information

We have seen that arbitrarily large representations of finite mathematical structures can be stored in random-access memory. If it were possible to build (at least in theory) arbitrarily large instances of RAM, then there would be nothing further to say. However, we saw that this is not the case. The storage hierarchy is not imposed by technological limitations, but is an inherent consequence of the physics of data storage: using matter which occupies space. Since the mass required must be asymptotically proportional to the amount of information being stored, and the density to which it can occupy space is ultimately limited, larger and larger amounts of information must take up more and more space (this should have been obvious from the start).

With these limits established, we obtain natural restrictions on the arrangement and distribution of data with respect to locations in memory:

- $O(1)$ information per location follows from bounded density
- $O(1)$ neighbors per location follows from finite dimensionality

The fact that matter cannot be compressed to an arbitrarily high density leads to the conclusion that at most one symbol of information can be stored in each location of memory. The bounded-degree requirement follows from the finite dimensionality of space, in that there are a fixed maximum number of adjacent locations that can be squeezed into a given radius. These local restrictions form what we will call *scalable* models of information, and correspond to memories of fixed word size and degree.

Homogeneity

We now turn our attention to a special uniform case in which all the models of the class are embedded in a single homogeneous background structure, one which looks the same everywhere. Although this may seem overly restrictive, this assumption does not detract from the import of our main result – it remains true with minor modifications, which are discussed at the end of the paper.

Our key objective is to define what it means for a class of finite models to be *uniform*. Essentially, this will be a class in which each finite model lives in a physically symmetric universe. This homogeneous *universe* \mathbf{U} consists of an infinite domain U of discrete *locations*, spatially related to one another by a finite number of *direction* functions $\Delta = \{\delta_1, \dots, \delta_d\}$, and arranged so that all locations “look alike”. Each location should be thought of as being equidistant from the d *neighbors* around it (to prevent overcrowding), and each direction is invertible, so that $\Delta^{-1} = \{\delta^{-1}: \delta \text{ in } \Delta\} = \Delta$. Any finite sequence of directions $\delta_1 \circ \dots \circ \delta_m$ can be composed, with the meaning first “go in direction δ_1 ” then... finally “go in direction δ_m ”. Note that adjacent inverses annihilate each other, and if directions were actual vectors in Euclidean space they would also commute, but we do not enforce this general requirement.

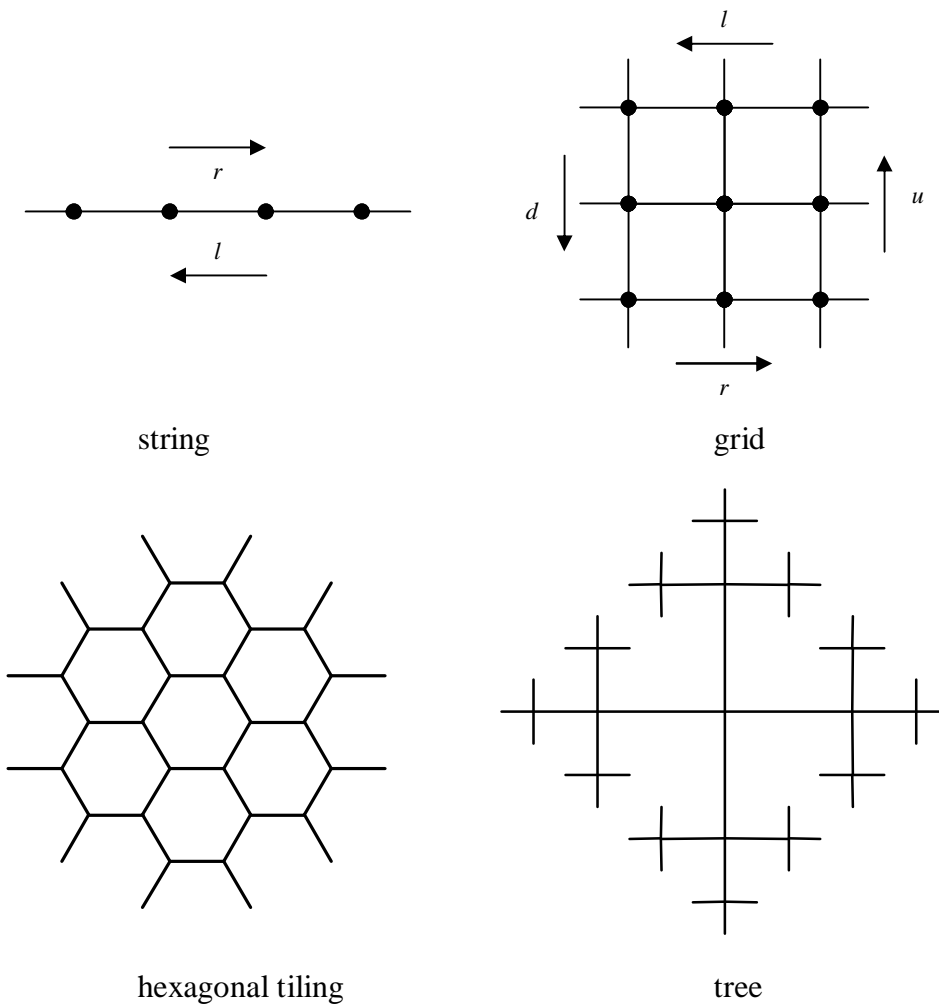
Indeed, arbitrary finite sequences from Δ , denoted Δ^* , form a group of bijections with respect to composition, acting on the elements of \mathbf{U} in a way that makes them completely indistinguishable from one another. This means that if a certain sequence in Δ^* makes a cycle at some location, it will make a cycle at every location.

Definition: A *universe* is an infinite countable domain U of discrete locations related to

each other by a uniform structure of direction functions δ_h . Specifically,

$\mathbf{U} = \langle U, \delta_1, \dots, \delta_d \rangle$ where each $\delta_h: U \rightarrow U$ is a bijection over the domain $|U|$ of locations, also satisfying *homogeneity*: for each sequence $\delta \in \Delta^*$, $\exists x \in U \delta(x) = x \leftrightarrow \forall x \in U \delta(x) = x$.

In particular, the set of cycles $C_U = \{\delta \in \Delta^* : \delta(x) = x\}$ is well-defined, and characterizes \mathbf{U} up to isomorphism (provided it is connected). A direct consequence is that each position has exactly the same number d of immediately adjacent positions, called its *neighbors* (provided of course that each δ_h is non-trivial). Typical examples of these constant-degree structures are conveniently visualized geometrically:



Drawings of a string, grid, hexagonal tiling, and tree – the last requires an infinite dimensional space in order to maintain fixed distances between elements.

Typically, when studying embedded finite models such as constraint databases, the

universe is a continuous structure such as the real numbers with ordinary arithmetic operations [Revesz], but our structures are discrete in order to maintain consistency with the “tape” used in theoretical models of computation. Nevertheless, it is useful to imagine our discrete universe of locations as living in a more physical continuous finite-dimensional metric space which is infinite in each dimension (so that the underlying algebraic structure becomes a torsion-free Abelian group). The most convincing examples are ones in which the elements are spaced equidistant in Euclidean space from their nearest neighbors, as illustrated.

Embedding models

These locations are thought of as potential places for storing information, using a finite amount of data in cells on an infinite tape. Hence, we insist that only a finite number of positions can hold symbols. For each symbol σ drawn from the tape alphabet Σ , we indicate its presence or absence at a given location by introducing a non-overlapping unary predicate (of the same name) to represent it, and adjoin all of them to the universe \mathbf{U} .

Definition: An *embedded finite model* is an expansion $M = (\mathbf{U}, \sigma_1^M \dots \sigma_s^M)$ of \mathbf{U} where the σ_i^M are pairwise disjoint finite subsets of U (we usually drop the superscripts), i.e. $\sigma_i \cap \sigma_j = \emptyset$ for $i \neq j$.

Although technically infinite, the model M contains only a finite amount of information, all within its *active domain* $A = \sigma_1 \cup \dots \cup \sigma_s$. These occupied positions are the only locations it makes sense to ask the contents of. Since unoccupied cells cannot contain any symbol (by definition) they are referred to as *empty*, and we will use a special *blank* symbol $\#$ later on to represent their locations. The adjacency relationship between cells of the active domain is inherited from the “spatial” relationship determined by the directions of the universe \mathbf{U} .

Data Structures

Restricting an embedded finite model to the contents of its active domain determines a finite relational data structure formed in the following natural way. The occupied cells become *nodes*, the symbols within them the *data*, with *edges* between them determined by directional adjacency. The relations $\{\sigma_i : 1 \leq i \leq s\}$ partition the active domain A , classifying each element according to its contents. Similarly, the functions $\{\delta_h : 1 \leq h \leq d\}$ assign a different type of edge to those pairs in A which are immediately adjacent in \mathbf{U} . Note that no two edges of the same kind can both point to or from any vertex. Instead, like edges are only incident from head to tail. More precisely,

Definition: To each embedded finite model $M = (\mathbf{U}, \sigma_1 \dots \sigma_s)$ associate the finite relational sub-structure (essentially a labeled graph)

$$D_M = (A, \sigma_1, \dots, \sigma_s, E_1, \dots, E_d) \quad \text{where } E_h = \{\langle x, \delta_h(x) \rangle : x, \delta_h(x) \in A\}.$$

In a certain precise sense, D_M (together with $C_{\mathbf{U}}$) contains all the finite information needed to reconstruct M up to isomorphism.

Uniformity

These embedded relational structures represent structured data which conforms to an underlying homogeneous universe. It is now quite simple to define what it means for a class of such data structures to be uniform: when all of them “fit” into the same fixed universal background structure. This is similar in spirit to the notion of ultra-homogeneous in [Hodges], but with only unary functions and relations.

Definition: A class of finite models K is called *uniform* if each one has the same background structure and alphabet. I.e. for a fixed \mathbf{U} and s , each model M in K is of the form $M = (\mathbf{U}, \sigma_1 \dots \sigma_s)$.

We now turn to the task of describing first-order properties of these uniform classes in logic.

5. Statement-Term Logic

In this section, we develop in detail a specially designed statement-term logic (STL) for describing properties of uniformly embedded finite models. Its key feature is that it has no variables in the conventional sense – rather the formulas refer directly to the very symbols themselves, as classes appearing on the tape *en masse*. It is important to note that STL utilizes the Subject-Predicate form of classical logic to effectively guard its quantifiers over the finite active domain. Yet despite this apparently strict limitation, it retains the full expressive power of first-order logic (FOL) with unrestricted quantification over the entire infinite domain. This active domain collapse was already observed by [Libkin] for structures of bounded-degree in the context of modern logic. Our contribution is demonstrating that it can be accomplished within the confines of a traditional logic applied to uniform classes of finite models.

Overview

It is important to recognize that our logical syntax will be bi-modal: *terms* representing classes of cells and *statements* indicating truth-values (possible configurations of the entire tape). Do not be dissuaded by its deceptive simplicity. It retains the full expressive power of modern first-order logic, provided attention is restricted to our spatially motivated uniform classes of finite models.

The idea is to be able to say things like “there are k elements, each of which resides in a certain kind of neighborhood”. In order to do talk about neighborhood types and their quantity, we must extend classical Aristotelian logic while retaining its monadic appearance. Two essential features are added: unary functions; and numerical quantification. Both additions maintain the essential character of a traditional approach to formal logic, and permit the sacrifice of polyadic relationships, including equality.

Although it is common to examine the deductive power of logic (what we can *prove*), we are concerned here with definability (what we can *express*). The main result is that our term logic is equivalent in expressive power to closed formulas of first-order logic. This is analogous to the inferential parity that term-functor logic enjoys compared to its modern counterpart [McIntosh]. For the reader who might want to explore the possibility of developing a deductive logic, a section on simplification rules is provided.

Terms

A *term* is a property of a cell which depends on its symbolic contents and that of its neighbors. Terms represent a set of places on the tape. They are built up from symbols in Σ and adjacencies in Δ by functional composition and connected with ‘and’, ‘or’, ‘not’ in a completely ordinary fashion. Symbols are undefined concepts as in formal language theory.

Symbols as Atomic terms

The vocabulary of our logical language begins with the finite *alphabet* Σ of input symbols which can appear on the tape (usually letters or numbers). The principle is to let the atomic term σ represent the class of places where that very symbol σ appears (that way both the term and the symbol have the same intention and extension). They sort of play the role of variables in modern logic (but not really).

Definition: Let each tape symbol σ in Σ be an *atomic term*, denoting all the places σ appears. Technically, $\sigma^M = \{u \in U : \text{the cell at position } u \text{ contains symbol } \sigma\}$.

Since by assumption only finitely many cells can be occupied by input symbols, atomic terms always denote finite classes. Also since no two input symbols can be coincident, and thereby occupy the same location, distinct atomic terms are disjoint. Note that the ‘blank’ is not considered a symbol. Rather, empty cells are discerned by the absence of every other symbol.

In a fundamental sense, symbols represent the actual information in the model. They correspond with associating to each σ in Σ the property “contains σ ” of cells.

Example: the atomic term **0** represents the places containing the symbol zero. This is indicated by the underlined positions in the string:

...###001011011###...

Directions for Simple terms

Augment the vocabulary by the fixed set of *directions* Δ that sets the adjacency relationships between cells on the tape. Functions are not entities unto themselves, only term modifiers. The principle is to use these functions to refer to the contents of nearby locations. Any bijection from Δ^* can be used to modify atomic terms via functional application.

Definition: For each δ in Δ^* and σ in Σ , let the *simple term* $\sigma\delta$ denote the class:

$$\delta[\sigma] = \{\delta(x) : x \text{ in } \sigma\},$$

i.e. the places which can “see” a σ by looking back in the direction δ^{-1} .

(Note that atomic terms are simple by choosing δ to be the identity).

Example: Letting l stand for “left of”, the term **1l** represents all cells which are to the left

of a one. This is indicated by the underlined positions in the string:

...###001011011###...

Note: Using postfix notation avoids the confusing transposition caused by function composition in ordinary prefix notation: $fgS = (f \circ g)S = g(f(S))$. Whereas even without parentheses, it is unambiguous to write $Sfg = S(f \circ g) = (Sf)g$.

Complementation

Given any property P it is often convenient to allow the complementary form non- P . These terms are written $\sim P$ and denote the places which do *not* satisfy the property P (may be infinite).

Definition: For any term P , let $\sim P$ denote the *complement* of P , which denotes $U \setminus P$.

Example: The term $\sim \mathbf{0}$ represents the locations which do not contain a zero. This is indicated by the underlined positions in the string:

...###001011011###...

Note: Parentheses are unnecessary for complements of simple terms because bijections pass right through: $(\sim \sigma)\delta = \sim(\sigma\delta)$.

Joining Terms

Terms can be joined with the connective operations of union and intersection.

Definition: Given terms S and T , both their *union* $S \cup T$ and *intersection* $S \cap T$ are terms.

It is easy to verify that bijections pass cleanly through them, so for any sequence of directions δ :

$$(S \circ T)\delta = (S\delta) \circ (T\delta) \quad \text{where } \circ \text{ is either } \cap \text{ or } \cup$$

Normalization of Terms

From the above observations, we can without loss of generality express arbitrary combinations of $\{\cap, \cup, \sim\}$ and bijections:

Fact: Any term can be written as a combination of monotone operators $\{\cap, \cup\}$ applied to simple terms or their opposites.

Proof: Push complements to the bottom with DeMorgan's laws, and distribute the bijections.

For example, referring to blank locations requires complementation. For the input alphabet $\{\mathbf{0}, \mathbf{1}\}$ blanks can be expressed as:

Example: The term $\sim(\mathbf{0} \cup \mathbf{1}) = \sim \mathbf{0} \cap \sim \mathbf{1}$ represents all empty cells, as indicated in the

underlined positions of:

...###001011011###...

To summarize, here is a table which inductively defines the syntax and semantics of terms.

<u>Terminology</u>	<u>Notation</u>	<u>Meaning</u>	<u>Conditions</u>
Atomic term	σ	$\{x : x \text{ contains } \sigma\}$	σ is a symbol in Σ
Simple term	$\sigma\delta$	$\delta[\sigma]$	δ is in Δ^*
Complementation	$\sim P$	$U \setminus P$	P is any term
Compound terms	$S \cup T / S \cap T$	union / intersection	S and T are terms

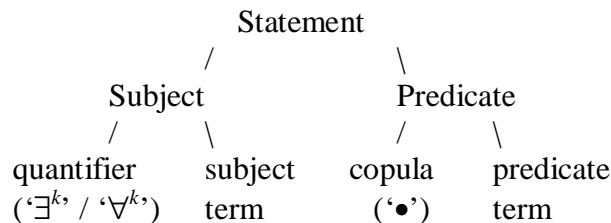
There is a difference between how we will interpret the semantics of terms, depending on whether they are used as subjects or predicates, though this distinction is not essential. A *subject term*, representing members of the active domain, is interpreted as a *class* or set of positions, so that for example, $\sigma \subseteq A$ refers to the set of all cells which contain the symbol σ . Since by assumption A is finite, these classes are finite also. A *predicate term*, on the other hand, is a Boolean-valued characteristic function on the universe U , mapping all locations (including empty ones) to $\{true, false\}$, so that for example $\langle \# \rangle(u) = true$ if and only if $u \in D$.

Statements

Unlike terms, which signify properties of individual things (i.e. cells in memory); statements signify facts about the state of the entire model (i.e. the contents of memory). The classical Subject-Predicate form provides a convenient way to include numerical quantity while still maintaining a finitely guarded domain of quantification.

Numerically quantified Subject-Predicate statements

The two-term syntax for statements in traditional logic provides no way to count. To remedy this limitation, we introduce a very natural extension in which quantifiers have numerical parameters in order to provide some sort of bounded counting capability, similar in spirit to [Murphree]. Here is the syntax diagram for the general subject-predicate form of our statements,



Compare the differences with modern logic, which relies on predicates applied to (functions of) variables. In particular, there are no atomic propositions in term logic following [Sommers p.5]. Instead, every statement asserts a categorical proposition about

how the subject term, according to its quantifier, satisfies the predicate term (the copula ‘•’ reads as “are”). For each parameter $k \geq 0$, they come in two forms, duals of each other.

Existential form: $\exists^k S \bullet P$ means “there are (at least) k S ’s which are P ”
Universal form: $\forall^k S \bullet P$ means “all but (at most) $k-1$ S ’s satisfy P ”

For $k = 1$ (which we often omit) these are the usual “*some* S is P ” and “*every* S is P ” respectively. The sentential negation “*no* S is P ” is given the usual treatment by converting it to term negation “every S is non- P ” (universals do not have existential import). For $k = 0$ these forms trivialize to the Boolean constants *true* and *false* respectively, even for a vacuous subject (which we do not prohibit).

Basic statements

Statements involving an atomic subject are intuitively elegant because they express categorical propositions about just one concept, even though the predicate term might be arbitrarily complex. This is the idea behind a basic statement.

Definition: If σ is an atomic term, and Q is either \exists or \forall , then $Q^k \sigma \bullet P$ is called a *basic statement*.

Example: $\exists^1 0 \bullet \#r$ means there is a left endpoint (in a string of all 0’s)

Figure: ...###00...0###... (the witness is underlined)

Moreover, we shall see later that requiring an atomic subject is not a significant restriction, in that arbitrarily complex statements can be reduced to combinations of just basic statements.

Sentential Negation

Despite numerical parameters, sentential denial still reduces to affirming the contrary with a reversal of quantity (the semantics were carefully chosen to make this happen):

$$\text{For } k \geq 0 \quad \neg \exists^k S \bullet P \Leftrightarrow \forall^k S \bullet \sim P \quad \& \quad \neg \forall^k S \bullet P \Leftrightarrow \exists^k S \bullet \sim P$$

as illustrated by:



So without loss of generality, negation can be eliminated so that all Boolean combinations of statements can be expressed by just using the positive sentential connectives of conjunction and disjunction.

Combining statements

Statements can be combined using ordinary propositional connectives.

Definition: If s and t are statements, then so are their *conjunction* $s \wedge t$ and *disjunction* $s \vee t$.

It is unnecessary to include negation as it can be ‘pushed through’ conjunction and disjunction.

In summary, here is a table which inductively defines the syntax and semantics for statements.

<u>Type of statement</u>	<u>Notation</u>	<u>Meaning</u>	<u>Conditions</u>
Basic existential	$\exists^k S \bullet P$	$ S \cap P \geq k$	S is an atomic, P is any term
Basic universal	$\forall^k S \bullet P$	$ S \setminus P < k$	S is an atomic, P is any term
Denial	$\neg p$	negation	p is any statement
Compound	$s \wedge t / s \vee t$	conjunct / disjunct	s and t are arbitrary statements

Call the set of all such statements **STL**, for **Statement-Term Logic**.

Example: $\forall^2 0 \bullet 0r \wedge \forall^2 0 \bullet 0l$ means there are no ‘gaps’ (in a string of all 0’s)

Figure: ...###00...0###...

Since any finite string has two “external” endpoints, it generates one exception for each clause. The formula says that there are not two exceptions, precluding any “internal” endpoints.

Simplification rules

This section is provided for the benefit of the reader who might be interested in seeing illustrations of syntactic transformations in STL. Although it can be safely skipped, do take note of the proposal on how to deal with the “head of an animal” problem. Note well that unnecessary parenthesis have been removed, so pay close attention to the distinctions between terms and statements.

Distributing over compound predicates

Just as in the non-numeric case, existential quantification distributes over union and universal quantification distributes over intersection, but it is more complicated because we have to split the quantity by numerical partitioning.

To distribute in the existential case, we must first separate the predicate into a *disjoint* union:

$$P \cup Q = (\sim P \cap Q) \cup (P \cap Q) \cup (P \cap \sim Q)$$

Now we can distribute over any two disjoint terms U and V :

$$\exists^k S \bullet U \cup V \Leftrightarrow \vee \{ \exists^i S \bullet U \wedge \exists^{k-i} S \bullet V : 0 \leq i \leq k \}$$

Similarly, in the universal case, use duality to first separate an intersection predicate into:

$$P \cap Q = (\sim P \cup Q) \cap (P \cup Q) \cap (P \cup \sim Q)$$

Now distribute over any two terms U and V whose union is everything:

$$\forall^k S \bullet U \cap V \Leftrightarrow \wedge \{ \forall^j S \bullet U \vee \forall^{k-j} S \bullet V : 0 \leq j \leq k \}$$

Breaking up compound subjects

In this section we show how a statement $Q^k S \bullet P$ containing a compound subject S (without complementation) can be broken down into a combination of basic statements. Reducing everything to atomic quantification gives further credence to the idea that subjects are tangible. Although this can be deduced from the corollary to our main result by a semantic argument, our syntactic treatment here will help illustrate an important way to deal with unique correspondences (among other things).

Intersections in the subject are easily dealt with by moving one of them into the predicate:

$$\begin{aligned} \exists^k (S \cap T) \bullet P &\Leftrightarrow \exists^k S \bullet T \cap P \\ \forall^k (S \cap T) \bullet P &\Leftrightarrow \forall^k S \bullet \sim T \cup P \end{aligned}$$

Unions can be broken up by quantifying over the pieces.

$$\begin{aligned} \exists^k (S \cup T) \bullet P &\Leftrightarrow \vee \{ \exists^i S \bullet P \wedge \exists^j T \bullet P \wedge \forall^{i+j-k+1} (S \cap T) \bullet \sim P : 1 \leq i, j \leq k \} \\ \forall^k (S \cup T) \bullet P &\Leftrightarrow \vee \{ \forall^i S \bullet P \wedge \forall^j T \bullet P \wedge \exists^{i+j-k-1} (S \cap T) \bullet \sim P : 1 \leq i, j \leq k \} \end{aligned}$$

Verifying these identities is straightforward via $|S \cup T| = |S| + |T| - |S \cap T|$.

To remove functions from the subject observe that bijections preserve cardinality via the one-one correspondence between terms $S \approx Sf$.

$$\text{For } Q \in \{ \exists, \forall \} \quad Q^k Sf \bullet P \Leftrightarrow Q^k S \bullet Pf^{-1} \quad \text{where } k \geq 0.$$

This observation allows us to formulate and solve the famous ‘head of an animal’ problem very simply, provided we assume a one-to-one correspondence between animals and their heads. In Subject-Predicate form, letting h stand for ‘head of’:

$$\begin{array}{lll} \text{Premise:} & \text{Every } \underline{H} \text{orse is an } \underline{A} \text{nimal:} & \forall H \bullet A \\ \text{Conclusion:} & \text{all } \underline{h} \text{eads of } \underline{H} \text{orses are } \underline{h} \text{eads of } \underline{A} \text{nimals} & \forall Hh \bullet Ah \end{array}$$

The conclusion is actually equivalent to the premise via the aforementioned transformation:

$$\forall Hh \bullet Ah \Leftrightarrow \forall H \bullet Ahh^{-1} \Leftrightarrow \forall H \bullet A$$

Put another way, $H \subseteq A \Leftrightarrow Hh \subseteq Ah$, which depends crucially on h being a correspondence.

It should be noted that to eliminate complement requires a more complex argument that uses finiteness. For the sake of completeness, we sketch how this could be done. Without loss of generality, assume it is applied directly to an atomic subject term by pushing all complements to the bottom and using the transformations above. Take $Q^k \sim \sigma \bullet P$ and change it by breaking $\sim \sigma$ into a disjoint union of all the other symbols in the alphabet together with the “blank” $\#$ (which has infinite extension). Applying the transformation for subjects joined by union repeatedly will result in a combination of statements in which the only non-basic statements will be of the form $Q^k \# \bullet P$ for some compound predicate P . Use duality so that we consider only the existential form $\exists^k \# \bullet P$. Distribute the quantifier over union so that P consists entirely of intersections between simple terms of the form $\sigma \delta$ and blank terms of the form $\# \delta$ (converting complements as before). If P contains any non-blank terms $\sigma \delta$, rewrite the whole formula to be centered around σ . Otherwise, if P consists entirely of blank terms (asserting that σ appears in direction δ with respect to the subject ‘#’), see that $Q^k \# \bullet P$ is equivalent to just *true* since there are infinitely many blank neighborhoods consistent with P .

6. Results

We show that statement-term logic is as powerful as first-order logic with equality in expressing properties of our embedded finite models. As a consequence of the proof, we will also obtain an atomic subject normal form for STL.

STL = FOL

Recall that in a model of information M , finitely many tape symbols reside in a fixed infinite background structure \mathbf{U} , naturally forming a finite model D_M within it. Furthermore, being uniform, there is a well-defined set of cycles $C_U = \{\delta \in \Delta^* : \delta(x) = x\}$. Both of these ideas will be used in the proof of the following result.

Theorem: Fix a universe \mathbf{U} . For each sentence ϕ expressible in FOL(=), first-order logic with equality, there is an equivalent ϕ' in STL, statement-term logic, and vice versa, so that ϕ and ϕ' agree on all models M embedded in \mathbf{U} :

$$M \models \phi \Leftrightarrow M \models \phi'$$

Proof: (STL \subseteq FOL) This is the easy direction because it is straight-forward to design a syntactic translation.

Simple terms in STL correspond to atomic propositions in FOL with one free variable in an obvious way:

<u>STL term</u>	<u>FOL formula</u>	the translation of simple terms for symbol $\sigma \in \Sigma$ and function $\delta \in \Delta^*$
$\sigma \delta$	\rightarrow	$\sigma(\delta^{-1}(x))$

Although compound terms are trivially dealt with as Boolean combinations of these, it is important to use a common free variable.

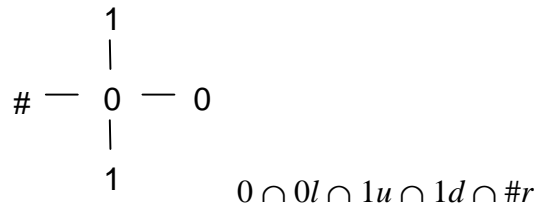
The only really interesting construction arises from numerical quantification, which relies on equality in a critical way. For the sake of a nice corollary, we translate existential statements in STL which have both arbitrary subject and predicate terms S and T , already translated by $\sigma(x)$ and $\tau(x)$ respectively, into FOL sentences:

STL statement FOL sentence translation of arbitrary statements, where S and T are any terms

$$\exists^k S \bullet T \quad \mapsto \quad \exists x_1 \dots x_k \sigma(x_1) \wedge \dots \wedge \sigma(x_k) \exists \{x_i \neq x_j : 1 \leq i \neq j \leq k\} \wedge \tau(x_1) \wedge \dots \wedge \tau(x_k)$$

Just take Boolean combinations of these (including negation to get \forall^k) in order to translate compound statements.

(FOL \subseteq STL): This is the hard direction because it requires a semantic analysis and construction. Perhaps the first to realize that first-order formulas can speak only about local properties of a model was [Gaifman]. But the easiest such normal form for our purposes comes from Hanf's Lemma, which for bounded-degree structures is most conveniently available from the treatment in [Immerman]. Basically, it says that on a bounded-degree structure, a first-order sentence is equivalent to a Boolean combination of sentences, each of which specifies a particular radius r neighborhood N along with the number of occurrences of N up to a certain threshold t . There are finitely many kinds of neighborhoods because r and t are determined by the sentence under consideration. Although Hanf's Lemma applies to purely relational structures, it is easy to make the appropriate modifications by inter-defining functions and relations. We will construct terms to describe these neighborhoods, and statements to specify how many times they occur.



a small example of a neighborhood in a grid and a term which describes it

A neighborhood N is characterized by an isomorphism type which specifies the exact shape of its positions (its structure) together with the contents of each position (its data), all with respect to a central location. By definition, its local structure is constrained by the embedding into our fixed global universe \mathbf{U} , so we can restrict attention to those isomorphism types whose shape is consistent with $C_{\mathbf{U}}$, for otherwise any nontrivial such existential assertion is automatically false. So it remains to completely specify the contents of each cell in N . Such a term becomes a property of the center by describing the arrangement of symbols around it using an intersection term $T_N = T_1 \cap \dots \cap T_n$, where each T_i is a simple term $\sigma\delta$ or $\#\delta$, using a variety of δ of length at most r . The quantity of

neighborhood types N appearing in the model M can be described by a combination of statements using numerical quantifiers with parameters k of magnitude at most t applied to T_N .

So all that remains to be shown is that $\Pi \equiv \exists^k T_N$ is reducible to a canonical (basic) statement with an atomic subject term. There are two cases. If some $T_i = \sigma\delta$, i.e. $T = \sigma\delta \cap T'$, then manipulate Π to be $\exists^k \sigma\delta \bullet T' = \exists^k \sigma \bullet T'\delta^{-1}$ which is of the correct form. Otherwise, every T_i is of the form $\sim\sigma\delta$, which entails $\#\delta$ (Recall that $\# = \sim\Sigma = \cap \{\sim\sigma : \sigma \in \Sigma\}$ stands for the blank, and that by definition completely empty neighborhoods can exist only outside the active domain.) Therefore T_N is consistent with a sufficiently large empty neighborhood. Since there are infinitely many of these in M , Π is equivalent to just plain *true* and we are done.

As a consequence of the last paragraph in the proof we obtain the following additional result.

Corollary: An arbitrarily complex statement of STL can be reduced to a combination of basic statements. I.e. subjects can be made atomic without loss of generality. In particular, this means that all quantification can be guarded over the finite active domain.

Remark: As promised, we indicate what would need to be changed to accommodate the non-homogeneous case, where we only assume that each node has bounded size and degree. Essentially, the main result would remain true except that we would have to describe local paths, adding special terms which predicate about whether or not a given sequence of directions forms a cycle. A term of the form ‘ δ ’ would mean all those positions for which traveling along the sequence labeled by δ returns one to the original position (in the homogeneous case, this would be trivially true or false, independent of the origin). In the interest of elegance and simplicity, we have chosen to present the homogeneous case only, because it eliminates the need for such paths. Otherwise, various simplifications that push functions across term constructors no longer hold. But with enough work, it is easy to see that the main result remains true.

7. Summary

We have analyzed scalability requirements for models of information by examining simple physical restrictions on storing arbitrarily large amounts of data in memory. The resultant classes of models are those whose structure takes into consideration the density of matter in space. The unlimited size of those structures forced us to acknowledge a bounded locality requirement for each piece of information stored.

From this natural perspective, we studied a simple but natural term logic based directly on the symbolic representation of information. This logic, motivated and inspired by the work of Fred Sommers, incorporates the classical subject-predicate form with no use of variables or indexing. Our main result was that despite its appearance, it is just as expressive as ordinary first-order logic over these feasible models.

The ultimate goal of this research is an examination of the role that physical resources place on the potential limits of computability. Therefore, a second paper will study the role of energy and time in models of computation.

Acknowledgements

First and foremost, I would like to thank my wife Suzanne for her many intellectual contributions to the exposition and ideas expressed here. Though sometimes an unwilling participant, her tireless encouragement and efforts on my behalf made this paper possible. I am also grateful to my dear colleague Scott Weinstein for innumerable conversations over the years which led me to this work, and for his careful reading of a draft of this paper. Also appreciated were the many stimulating conversations with Sheila Greibach, Yiannis Moschovakis, Wilfried Sieg, Yuri Gurevich, and Leonid Levin. A special thanks goes out to my departmental colleague David Wonnacott for his penetrating insights into the nature of computer science and its relationship to both mathematics and physics. In addition, I would like to thank my son Natan for expertly drawing some of the figures.

References

- Church, Alonzo *Introduction to Mathematical Logic*, Princeton University Press © 1956.
- Enderton, Herbert *A Mathematical Introduction to Logic*, Academic Press © 2001.
- Feynman, Richard “There's Plenty of Room at the Bottom” December 29th 1959 at the annual meeting of the American Physical Society, California Institute of Technology. Transcript: February 1960 issue of Caltech's Engineering and Science, available at <http://www.zyvex.com/nanotech/feynman.html>.
- Gaifman, Haim “On local and non-local properties” *Logic Colloquium '81* (J. Stern, editor), North Holland © 1982.
- Hamming, Richard *Coding and Information theory*, 2nd edition, Prentice-Hall © 1986
- Hartmanis, Juris Turing Award Lecture: “On Computational Complexity and the Nature of Computer Science” *ACM Computing Surveys* 27(1): 7-16 (1995).
- Hodges, Wilfred *A Shorter Model Theory*, Cambridge University Press © 1997
- Immerman, Neil *Descriptive Complexity*, Graduate texts in Computer Science, Springer © 1999.
- Libkin, Leonid “A collapse result for constraint queries over structures of small degree” *Information Processing Letters*, 86 (2003), 277-281.
- McIntosh, Clifton Appendix F of [Sommers]
- Murphree, Wallace *Numerically Exeptive Logic: A Reduction of the Classical Syllogism*, American University Studies, Series V, Philosophy, Vol. 112, Peter Lang © 1991.
- Revesz, Peter *Introduction to Constraint Databases*, Springer Verlag © 2002.
- Sommers, Fred *The Logic of Natural Language*, Oxford Clarendon Press © 1982.
- Toigo, Jon William “Avoiding a Data Crunch” *Scientific American*, May 2000, pp. 58-74.
- Turing, Alan “Intelligent Machinery” in *Collected Works of A.M. Turing: Mechanical Intelligence* edited by D.C. Ince, North-Holland © 1992.
- Ullman, Jeffrey *Principles of Database Systems*, 2nd ed., Computer Science Press © 1982.