Signal Space, Address Space, & Symbol Space

James C. Brakefield
Technology Incorporated, Life Sciences Division
300 Breesport, San Antonio, Texas 78216

Abstract:

Three major engineering disciplines (computer or hardware, software, and knowledge) can be abstracted so that they deal with vector spaces. The three abstract spaces can be mapped into one another such that concepts in one take another form in one of the others. This leads to insights into the nature of various problems.

Boolean signal space is a variation of the binary hyper-cube. It can be used as a framework for logic design.

Address space is a variation of Forth. It can be used as a framework for the programming of computers.

Symbol space is an approach to the artificial intelligence paradigm. Its intuitive appeal is as a model of thought.

The talk will describe each of these in greater detail, show the mappings from one to another, and show some problems which benefit from the recasting of their context.

A. Binary Signal Space

Binary signal space is a collection of binary inputs, outputs, and intermediate signals considered as a whole. Usually each is considered an orthogonal axis giving rise to the hyper-cube. This is the usual starting point for logic circuit design. My concern is with the entire system, such as a complete computer with all its memory modeled as gates. Thus the area of interest is thousands, millions, or even billions of signals.

Some questions can be asked in the presence of such large numbers: Is it possible to fully specify the complete truth table with so many signals? How does one know a particular circuit is correct? What happens to the circuit as truth table entries are added? What happens to the circuit when signals are added?

My interest is mapping the pattern recognition and learning situations into logic design. Each pattern or learning example is a single truth table entry. Any mechanism that "learns" is then doing circuit design where entries and/or signals are added one at a time to the truth table. The truth table entries can be thought of as labels of the veriticies of the hyper-cube.

In such a circuit design environment the truth table is extremely sparse. With only one hundred signals there are 2**100 or 10**30 entries. Even a million entries (say 10 by 10 binary raster images) is an insignificant portion of the hyperspace. These unfilled entries are essentially "don't cares". The circuit design software or hardware assigns them so as to simplify the circuit.

B. Address Space

Address space is an abstraction of the programming paradigm. Forth is very suitable for this. Programs consist of address strings referencing either other address strings or primitives. The interpretation of the strings is sequential whereas the signal space merely computes a function.

A feature common to both address space and signal space is decomposition. There is a hierarchy of gates and subroutines. If a subroutine calls itself directly or indirectly one has recursion. If a gate is fed by itself directly or indirectly one has a feedback loop.

A pure functional approach will map recursion and feedback into an infinite tree structure. This gets rid of the loops by macro expansion. The other approach is to name at least one signal or address in a recursion or feedback loop so in one's mind the loop is broken.

It is in some sense most important to know the decomposition structure. The really "hard" part of programming is choosing an appropriate decomposition of the problem. Likewise in circuit design.

There is little said about the expression of decomposition structure. However, it is critical. Modern circuit design is hierarchial and structured witnessed by the acceptance of CAD systems using this approach. All the more "advanced" languages (Forth, Lisp, Prolog) support tree structures both as program and as data.

Another term used is "factoring". If one can consider a hardware circuit description language or a programming language as an algebra, then the structure of a program or circuit is the factoring of a problem into "terms". The real advantage of Forth is the ease and generality of its factoring ability.

There appears to be a duality between hardware and software when expressed as signal space and address space. Feedback circuits form the backbone of memory and are the most prominent signals to have names. Recursion loops in address space are usually broken by naming, thus there is a duality between feedback and recursion. In the signal space implementation of address space the feedback loops are enumerated via address decoders. These then become the "units" of the address space. In the address space implementation of signal space the recursion of routines calling one another is broken by the symbol table which enumerates the signals.

The dual of time is space. Thus the hardware operates all signals in parallel over time. Software operates signals sequentially in time with memory being in parallel over space.

I make the case that much of human "learning" is the search for such decompositions or structure. That this is the expression of generalization and knowledge. One can even make the case that this is the core of philosophy: Abstract philosophy is the study of decomposition in and of itself. Applied philosophy is the study of decompositions occurring in certain areas or fields. Science is the search for decomposition structures in reproducible phenomena. Engineering is the application of known decomposition structures.

C. Symbol Space

Symbol space is the hoped-for algebra of thought. We all know what symbols are. They are the tokens representing our experiences. I.e., the intuitive meaning of a symbol is what is considered a single thought. The progress of civilization is measured in our ability to make real these thoughts. Thus speech, writing, and graphics. Symbol processing is the basis of language. Two symbol spaces are similar when they can communicate. The implications are that a government must ensure similar "symbol spaces" in its citizens, and thus the real purpose of our educational institutions.

The Turing test can be rephrased as the task of building a symbol space equivalent to that of an average person. I think this phrasing gives a better understanding of the problem of simulating human conversation.

Most of the programming languages used in AI work allow structure to be expressed easily, the dictionary of Forth, the CAR-CDR lists of LISP, and terms of Prolog. These are all ways of creating and naming tree structures. They are all also executable in some sense.

D. Mappings

Since computers, the mechanism implementing address space is built from gates, signal space can implement address space. The major technique is the enumeration of a large number of feedback circuits, i.e., memory. This enumeration is implemented by address decoders and associated read and write circuits. The cost of these auxiliary circuits is such that usually only a single path to memory is provided, and hence the von Neuman bottleneck.

When address space implements signal space it is called a logic simulator. Interestingly enough the Forth code string representation of a logic circuit is about the same as the data structures used, in say, a Fortran-based logic simulator, i.e., there is a code string for each gate which when evaluated/interpreted updates the output of the gate.

The net result is that a subset of the memory locations represent the state of the logic circuit. This is the dual of the logic circuit representing a computer where a subset of the gates represent the memory contents.

Symbol space is a set of symbols and their algebra. In address space symbols are memory locations, and their algebra is the executable tree structures they reference. In signal space the symbols are signals and their algebra is their effect on the rest of the circuit. Thus, a symbol space can be implemented by either signal space or address space. So far symbol spaces have been implemented as programs and the programs implemented by computers. This need not be so, the compilation of silicon being the means for going directly from specification to hardware.

A given signal space or address space is finite. Thus the symbol spaces either can simulate will also be finite. This can be expected to give rise to an interesting set of aliasing or truncation phenomena of which the next several decades will present many examples. The most common expression of this is the limited context which AI programs are able to use, i.e., their lack of common sense.

There are several relations between all three: If we take Prolog as a means of expressing things in symbol space, then translate this into signal or address space: The clauses of Prolog can be directly made into Forth code strings or emulated by a hardware logic simulator. The clauses are also a means of specifying a truth table.

E. Summary

Much as physics gives rise to several engineering disciplines (Mechanical, Civil, Industrial) so may what is legitimately called computer science gives rise to computer, software, and knowledge engineering. It is still a science in its infancy whose boundaries and areas of application are unresolved. A better name might be the science of information structure.

Much exploration of symbol algebras needs to and will be done. The mere idea of what constitutes computation is being debated. The duality between signal space and address space offers opportunity for a more scientific analysis of the trade-offs between the two. The effects of finiteness of the implementation of symbol spaces needs analytical treatment.