Situation Assessment in Computational Semiotics

Christopher Landauer
+1 (310) 336-1361, cal@aero.org
Kirstie L. Bellman, Principal Director
+1 (310) 336-2191, bellman@aero.org
Aerospace Integration Science Center, Mail Stop M6/214
The Aerospace Corporation
P.O. Box 92957, Los Angeles, California 90009-2957

Contents

Abstract

This paper is about a particular technical problem in the study of autonomy in computational systems: how can a computing system invent and use its own symbols and symbol construction mechanisms?

We think this invention is necessary for intelligent behavior in any organism, artificial or otherwise, since it has been observed repeatedly that systems with fixed symbol sets get stuck. This observation is well-known in computing circles, and has been observed in "extensible" languages, large and growing knowledge bases, content-based filing systems for technical literature, and inheritance hierarchies for large object-oriented systems: no matter what elaboration mechanisms are used, no matter what initial structures are used, the structures eventually become too constrained by what is already there, and stagnate. To change that phenomenon, we need construction mechanisms that can themselves be changed and elaborated.

We think that if we can change the construction mechanisms (which include the alphabet, as the mechanisms for constructing expressions from nothing), that is, the basic units that combine or are combined to make the structures, then we have another way to escape the trap of creeping rigidity: we can elaborate the units out from underneath the existing structures. This paper is a first step towards this technical problem; it is primarily intended to stimulate discussion and further research in these areas, since the questions are very hard and have no easy answers.

1 Getting Stuck in Conceptual Spaces

A *Constructed Complex System* that we expect to act autonomously will have to account for various different relationships among entities in its environment [1] [8] [9] [2]. It will need to express a relationship, discover or identify it "in the wild", distinguish it from others, record information about it, and even identify instances of it. These expressions and refinements of them become the basic tools of symbolic reasoning.

For example, consider Knowledge-Based Systems and ordinary computer programs. We start with a fixed set of types or type constructing mechanisms (all of the types are finitely consructed). The program can build new objects and relations. It can even build new types, but it cannot build new type constructing mechanisms. On the other hand, we, as developers, can add new mechanisms. In this case, each new type and each new type construcing mechanism has a relationship with the old ones.

We are interested in the cause of the frequent problem that the relationships become denser and more rigid as more and more concepts and relations are added. This is what we call the "get stuck" problem. It has something to do with the conceptual density of the existing structure, and its relationship with the intended problem domain, but it is not a solved problem or even a well-posed one yet. We formulate one version of the problem and show how it arises.

We make a series of assumptions about how these autonomous systems can or must operate, and then prove two simple limiting theorems. However, we do not believe that these theorems represent a fundamental limitation on computing. Rather, since

A failure of imagination is not a proof of impossibility (C. Landauer),

and since we would prefer that computing systems not be so limited, we examine the assumptions in turn. In our opinion, the most limiting of these assumptions is about using fixed sets of basic symbols and symbol expression constructors. We show that a system with a fixed basic symbol set will get stuck, so we need to consider computing systems that can change their symbols. This paper is mainly intended to highlight this particular technical problem in the study of autonomy in computational systems:

How can a computing system invent and use its own symbols?

This problem is especially difficult because computational systems generally have their symbols grounded in a small discrete system such as ASCII. We want to construct complex heterogeneous computing systems that have increasingly interesting and robust complexities of behavior, and we believe that this goal requires an advance in computational practice: changing the ways in which symbols are represented and used in computers. We must therefore define computationally viable theories of symbols and representation, that is, ways for programs to work more effectively with the meanings that underlie the labels used in them, given the relatively primitive operations on those labels currently available to computing systems. We believe that these theories cannot be restricted to structures built from indivisible units, and that we therefore need some new developments in Mathematics to treat this new kind of semiotic foundation for systems [6].

2 On Getting Stuck

The first step along these lines is what we call the "get-stuck" theorem, a result that is well-known in computing circles, at least operationally, and has been observed in "extensible" languages, large and growing knowledge bases, and inheritance hierarchies for large object-oriented systems: no matter what fixed finite set of elaboration mechanisms are used, no matter what finite set of initial structures are used, the constructed structures eventually become constrained too much by what is already there, and stagnate, or at least become extremely difficult to extend further. To change that phenomenon, which is a kind of creeping rigidity of partial success, we need construction mechanisms that can themselves be changed and elaborated.

There are actually two kinds of "get stuck" theorems. The first ones show that there is a limit on how finely a system can discriminate using a fixed symbolic system, and the second ones show that there is a limit on how finely a system can be elaborated using a fixed symbolic system, even if humans are doing the elaboration.

Our notion of the first kind of theorem is that even though we seem to be able to expand (tree-structured) hierarchies indefinitely (e.g., directory structures, ordinary ontologies, context-free grammars), they cannot express some of the important connections between different kinds of knowledge; it is the cross-connections that give us much richer expressive mechanisms (e.g., directed graphs context-sensitive or even general phrase-structure grammars). We "get stuck" in our attempts to express important aspects of our complex environments.

Our notion of the second kind of theorem is that when we use these more interesting structures, we eventually "get stuck" in our attempts to extend them.

2.1 First Theorem

We appear to live in a world of continuous time and continuous space, both of which are effectively infinite in extent and in resolution (quantum mechanics to the contrary notwithstanding). The space has a kind of local smoothness, but it is not uniform, since bifurcations do occur.

We make the following assumptions for the sake of the theorem: the system we consider is placed in an environment. It uses symbolic constructs to discriminate situations that matter. It has a maximum computational speed, and therefore the size of the symbolic constructs also matters.

We define some measurements of capability and expressive power of an autonomous system's symbolic mechanisms.

The general problem is to find a way to describe the environmental complexity, and the complexity required of an autonomous system that we expect to survive in that environment. We use the terms "density" and "fraction", because they indicate a kind of filling of the available conceptual space, but we do not assume that density is computed as a simple fraction of what is actual compared to what is possible, since that particular way to measure will not always be useful (especially if there is any kind of fractal behavior).

The relevant new terms are *expressive density*, which is what fraction of situations can be expressed, *expressive power*, which is how much effort it is to express them, *discriminative density*, which is what fraction of situations can be distinguished from others, *discriminative power*, which is how much effort it is to distinguish them, *situational density*, which is what fraction of differences in the situation matter and *situational power*, which is how much effort it is to recognize them.

In this context, there is a lower bound on effectively representable resolution: any fixed symbol system has a lower bound on units and an upper bound on speed, hence a lower bound on time intervals and a lower bound on space-time-behavior resolution.

The expressive density of a symbolic system in a discrete environment is some comparison of

> the number of structures of "size" n

and

> the number of possible regions to express (in d-space, we have d-cube stacks, so the number grows with a power d).

For example, for a context-free grammar, we let $s(n)$ be the number of sentences of length n (that is, the number of strings of length n that are consistent with the grammar), and ask about

$$s(n) \; / \; |A|^n$$

as n gets arbitrarily large, (throughout this paper, we use the expression $|A|$ for the size of the set A). or even

$$\log s(n) \; / \; n \; \log |A|.$$

For situational density, we need to determine what are the situations that matter. Then the expressive density does not need to be more than that, nor does discriminative density Both measures depend on the geometry in the situation space: two close together situations matter equally, and two close enough situations cannot be distinguished.

Another complexity notion is Ogden's Lemma [3], which essentially says that longer and longer strings in a context-free language must begin to show regularities, and must in fact come from repeated substrings. The reason that this is a complexity notion is that it constrains the kinds of strings that can be defined by a context-free grammar, and since there are many strings that cannot, the expressive density of a context-free grammar gets less and less complete as the size gets large.

We do not think that complexity notions like these let us prove that purely symbolic mechanisms must fail or that we must use some dynamics, because there are other assumptions that can be changed, as we shall show.

The first theorem is that fixed construction methods are greatly limited in how much they can express and distinguish. This theorem is about the complexity of environment that can be distinguished or expressed by structures built according to these rules. We need to note what situations are and how they can be represented, identified, and distinguished. We are trying to build *constructed complex systems* that reason symbolically about their environments (i.e., using some kind of discrete symbols).

First we describe the expressive mechanisms in sufficient detail that we can state and prove our first theorem.

The mathematical tool we use is the application of formal power series called *generating functions* [12], which are commonly used for combinatoric problems, since they allow us to deal analytically with large sequences of numbers.

Assume that we have a fixed finite set S of construction methods, an arity $n(s)$ for each s in S (the *arity* of a combination method is the number of formal parameters it needs, so for example, numerical constants have arity 0 and division has arity 2). Construction symbols that allow multiple arities are simply represented into different constructors with each of their arities.

There are clearly some restrictions in these assumptions, and we make no claim that every expressive mechanism can be described this way (indefinitely accumulative operations like sum and product do not fit these constraints, for example). Certainly any that can be parsed using context-free grammars into trees with bounded branching factor and a finite number of different node types can be described this way, and that is a large enough class of expressive mechanisms to make the theorems interesting.

Assume also that we have a *valuation* v, which assigns a positive integer cost or value $v(s)$ for each s in S. The simplest example is to take $v(s) = 1$ for all $s \in S$, which is the size of the individual symbol.

We extend the valuation v to all structures in the usual way:

$$v(s(u_1, u_2, ..., u_n)) = v(s) + \sum(\{i : 1 \leq i \leq n\}) v(u_i),$$

where $n = n(s)$. This just means that the cost of an expression is the sum of the costs of its constituents, plus the cost of the construction method itself (i.e., the whole costs $v(s)$ more than the sum of its parts). For example, if $v(s) = 1$ for all s, then $v(c)$ for any structure c is the size of c, excluding the parentheses.

The constructors for which $n(s) = 0$ are the constants (the basic units from which all structures are made). There are typically a very small number of these (e.g., 10, 26, 255 for some well-known sets).

For any composite structure c, we write that the *type* of the structure c is s, or

$$t(c) = s,$$

when the outermost construction method in c is s in S. Because all constructs c are built using one or more of the construction methods, every construct c has a unique type.

If we then define the *generating function* $f_s(x)$ for s in S as the formal power series

$$f_s(x) = \sum(\{c : t(c) = s\}) x^{v(c)},$$

which is the same as

$$f_s(x) = \sum(n \geq 0)|\{c : t(c) = s, v(c) = n\}|x^n,$$

then we can show that each $f_s(x)$ is an algebraic function (that is, it satisfies a polynomial equation), which is a refinement of an old theorem of Chomsky and Schu:tzenberger [3] [11]. The fact that these power series are algebraic limits the rate of growth of their coefficients, thereby limiting the number of composite structures of any given size.

The general claim is that we can discriminate more situations more finely by using larger and larger description structures, but the limited number of structures of each size means that the structure size grows very fast relative to the number of situations to be discriminated. There eventually comes a point at which the time and cost of description or processing is too large for any useful response to be computed in the time available. That point is a limitation on the expressive and computational mechanisms; they can't do any more any faster.

To avoid this problem, which is the first "get-stuck" problem, we have to allow the expressive mechanisms to contain cross-connections. We show in the next section that they do not solve the problem for us. They lead to the second "get-stuck" problem.

2.2 Second Theorem

The second "get-stuck" problem is about using expressive mechanisms of the form we have described to organize conceptual domains. To avoid the first "get-stuck" problem of expressive power, we must allow cross-connections, which means that context-free grammars are no longer sufficient. In any rich domain, as the extent of the description gets larger, and as the amount of detail in the descriptions gets larger, the cross-connections begin to dominate: it becomes harder and harder to make changes to the overall conceptual structure. There are more interconnections among existing parts, so changing one causes or at least requires changes in many.

If the system uses structures to discriminate situations in the environment, it will need more and more structures for finer and finer discrimination. However, having more structures implies making it more cumbersome to change: there are more interconnections, more obscure interconnections, more surprises, and more unforeseen implications. This is the creeping rigidity of partial success, which does not prevent autonomy, it merely makes it harder to implement, since the first theorem shows that then there are limitations to the expressive power, and hence to the ability of the system to describe and manage its environment.

These expressive mechanisms, even when designed to be extensible, seem to get stuck by over-constraining the possible refinements, even when the refinements are organized and implemented by humans (in other words, this is not simply a Turing or Go:del theorem about what is or is not computable or decideable by algorithms). We think that if the system can retain the fluidity in the original structures, that is, if the system can replace the basic symbolic units and the corresponding construction mechanisms, this problem might go away, at least partially.

So, for example, consider a directed graph, which represents the influential connections among nodes, and assume multiple interconnections among a new node and the existing ones (if there is only one new interconnection, then we have a tree and the first theorem applies, which is that the structure is not rich enough to express what we think will need to be expressed).

In any bounded domain, if when the current density of concepts is d, then a new node connects to (or to some random number of nodes with expected value) at least $c.d$ other nodes for some constant $c > 0$, then n nodes have (or are expected to have) at least $c.d.n.(n-1)$ / 2 connections, out of a possible $n.(n-1)$ / 2, so the ratio is constant. In particular, since the density increases, the number of connections to new nodes increases. This is the essence of the theorem. We can make a number of refinements of the theorem to use local densities, random new connections, or some other notion of density (in our statement above, we did use density as a fraction of the total), but the conclusion remains the same.

Another aspect of the theorem is that when the domain is widened (which we can see will reduce the density and would hope would change the results), there are many nodes that were previously part of the context that now need to be expressed explicitly, so that the density may in fact not decrease at all, and even if it does, it is one-time only, and the result of the theorem holds with a different constant c. Either way, rigidity increases.

An immediate implication of this result is the difficulty in "scaling up" of any conceptual structure defined in symbols. Just as in any building project, and as is well-known in software and system development projects, the conceptual infrastructure required for very large conceptual structures is quite different from that required for medium and small ones. It is not possible simply to extend a small one until it becomes large.

3 Symbolic Systems

The second theorem implies that the system must be able to change the construction mechanisms in order to escape the rigidity trap. An alternative interpretation is that if we can change the alphabet, that is, the basic units that are combined to make the structures, then there is another way to escape the trap of creeping rigidity: we can elaborate the units out from underneath the existing structures. This brings us directly to Computational Semiotics, which for us is about the treatment and use of symbols in computing systems, or equivalently, about how to rearrange computational elements along semiotic principles [5] [6]. The practical problem with this interpretation, of course, is that we are trying to build a system whose grounding can be arbitrarily deepened in a computing environment that is rigidly grounded in bits and bytes. It presents some interesting challenges.

Our particular approach to a solution requires having a system that can decide for itself that it needs to change its own foundations, by inventing new baisc units and the corresponding combination methods, and that can subsequently carry out the change, by mapping all of its current structures into the new foundation.

So the two operative questions in this approach are:

> How does a program decide when to invent new symbols?

and

> How does a program invent new symbols?

For the first question, it must determine in some way that it needs to make new distinctions. For the second question, it must have a way to recognize which expressive mechanisms need refining, which new distinctions they must support, what new computational processes need to be made available, and how the expressive mechanisms need to be changed to do so.

New distinctions, refined expressions, and new computational processes are all part of the elaboration of structure. We claim that the driving force behind the elaboration of structure is competition, since we consider cooperation to be a way to build larger units for competition at different scales. Actually, both terms reflect the same axis of interaction: they are about entities building models of other entities' behaviors and using those models in their own prediction- and decision-making processes. For example, our own conceptual refinements result from a competition between our models and the explanations or other behaviors that we demand from them. Our computational refinements result from a competition between the computing systems and the operations we require of them in the environments in which we place them. The problem in the latter two cases is that we must be the agents of change. What we actually want to do is to allow the computing system itself to be the agent of change.

We think that moving to a meta-level description will help in this case, since a finitely constructed description can describe an infinite object (e.g., a context-free grammar is finite, yet it can easily be used to describe an infinite language). If we consider a finite description of an expression construction method, then we can easily imagine that the construction method need not be a finite symbol system of the kind that occur in our "get-stuck" theorems.

4 Conceptual Categories

Finally, we describe our conceptual categories [4], because they are a kind of representational mechanism that we believe has a much better chance of being able to avoid the rigidity trap than most others.

We start by separating the "category" itself from the various structures that it may have, or that elements of it may have. These structures are called "divisions", mainly for historical reasons, and they are only defined for a category relative to an explicitly defined "viewpoint", which has a "context", a "scope" and a "focus". The scope defines the extent of the division, the focus defines the level of detail of the division, and the context defines a collection of interpreters for the symbols used in defining the scope, focus, and division.

There may be multipe viewpoints, with different divisions according to different viewpoints, with different membership criteria, and even with no obvious relationship between the same objects in different viewpoints.

These categories can be used to represent many kinds of knowledge, including modeling decisions. First, the "thing"ness of all linguistic constructs means that we can represent categories of objects, processes, qualities, relationships, and anything else. In order to have the process descriptions actually *do* something, we only need to find a mechanism for going from the "declarative" descriptions in the divisions of the category to the "procedural" operation of their behavior. Relationships are easier: a relationship is a division of a category, which has explicit *roles* for other categories. This separation of roles from the models that may fill them is one of our research results [7], derived originally for simulation applications.

The first reason why we expect that this mechanism will help us avoid getting stuck is the separation of the categories and the divisions, which means that other divisions can be added without affecting old ones, and that multiple viewpoints can coexist. The second reason is that any and all structuring mechanisms can be used as divisions, so a system has some chance of being able to represent a very large number of different kinds of relationships. The third reason is that the categories can be refined without changing existing structures, so that the system can retain old computational methods and also make new ones.

5 Conclusions and Future

Any computer program has a finite amount of code available to it, in the original program source code, the compiler, and the run-time system.

The two "get-stuck" theorems we proved in this paper show that either we have to abandon fixed symbol systems, or else we have to abandon our expectation that a system can refine its representations indefinitely, until their complexity is adequate to describe the expected environment.

Both theorems need to be extended to less stringent hypotheses, and the questions of Section 3 answered, especially the one about the effects of moving to the meta-level.

References

[1] Kirstie L. Bellman, Christopher Landauer, "A Note on Improving the Capabilities of Software Agents", pp. 512-513 in *Proceedings AA'97: The First International Conference on Autonomous Agents*, 5-8 February 1997, Marina Del Rey (1997)

[2] Kirstie L. Bellman, Christopher Landauer, "Computational Embodiment: Biological Considerations", pp. 422-427 in A. M. Meystel (ed.), *Proceedings of ISAS'97: The 1997 International Conference on Intelligent Systems and Semiotics: A Learning Perspective*, 22-25 September 1997, NIST, Gaithersburg, Maryland (1997)

[3] J. Berstel, L. Boasson, "Context-Free Languages", Chapter 2, pp. 59-102 in [10]

[4] Christopher Landauer, "Conceptual Categories as Knowledge Structures", pp. 44-49 in A. M. Meystel (ed.), *Proceedings of ISAS'97: The 1997 International Conference on Intelligent Systems and Semiotics: A Learning Perspective*, 22-25 September 1997, NIST, Gaithersburg, Maryland (1997)

[5] Christopher Landauer, Kirstie L. Bellman, "Semiotics of Constructed Complex Systems", pp. 35-40 in *Intelligent Systems: A Semiotic Perspective, Proceedings of the 1996 International Multidisciplinary Conference, Volume I: Theoretical Semiotics, Workshop on Intelligence in Constructed Complex Systems*, 20-23 October 1996, Gaithersburg, Maryland (1996)

[6] Christopher Landauer, Kirstie L. Bellman, "Mathematics and Linguistics", pp. 153-158 in *Intelligent Systems: A Semiotic Perspective, Proceedings of the 1996 International Multidisciplinary Conference, Volume I: Theoretical Semiotics, Workshop on New Mathematical Foundations for Computer Science*, 20-23 October 1996, Gaithersburg, Maryland (1996)

[7] Christopher Landauer, Kirstie L. Bellman, "Model-Based Simulation Design with Wrappings", pp. 169-174 in *Proceedings of OOS'97: Object Oriented Simulation Conference, WMC'97: 1997 SCS Western Multi-Conference*, 12-15 January, Phoenix, SCS International (1997)

[8] Christopher Landauer, Kirstie L. Bellman, "Computational Embodiment: Constructing Autonomous Software Systems", pp. 42-54 in Judith A. Lombardi (ed.), *Continuing the Conversation: Dialogues in Cybernetics, Volume I, Proceedings of the 1997 ASC Conference*, American Society for Cybernetics, 8-12 March 1997, U. Illinois (1997)

[9] Christopher Landauer, Kirstie L. Bellman, "Computational Embodiment: Software Architectures", pp. 205-210 in A. M. Meystel (ed.), *Proceedings of ISAS'97: The 1997 International Conference on Intelligent Systems and Semiotics: A Learning Perspective*, 22-25 September 1997, NIST, Gaithersburg, Maryland (1997)

[10] Jan van Leeuwen (Managing Editor), *Handbook of Theoretical Computer Science, Vol. B: Formal Methods and Semantics*, MIT (1990)

[11] A. Salomaa, "Formal Languages and Power Series", Chapter 3, pp. 103-132 in [10]

[12] Herbert S. Wilf, *generatingfunctionology*, Academic Press (1990)