Semiotic Morphisms, Representations and Blending for Interface Design

Joseph Goguen*
Department of Computer Science & Engineering
University of California at San Diego, USA
jgoguen@ucsd.edu

Abstract

Issues of representation arise in natural language processing, user interface design, art, and indeed, communication with any medium. This paper addresses such issues using algebraic semiotics, which draws on algebraic specification to give (among other things) an algebraic theory of representation, and a generalization of blending in the sense of cognitive linguistics. New ideas in this paper include distinguishing structural and conceptual blending, using co-relations for blending, and using hidden algebra for dynamic signs. Some examples are developed in detail, including their formalization in the BOBJ language.

1 Introduction

Notions of design, representation and meaning in formal computer science are impoverished compared with what is necessary for applications like web design and natural language understanding. However, these notions are precise, amenable to implementation, and have many applications. This paper describes *algebraic semiotics*, a formal theory of complex signs addressing interface issues, in a general sense of "interface" that includes user interface design, natural language, and even art. Algebraic semiotics is broader than most computer science, but still precise and implementable; it is based on concepts from algebraic semantics and cognitive linguistics, as briefly reviewed in Section 2. BOBJ is used to give formal specifications for representations and blends, and all code here has been run. An advantage of formalization is that, by forcing one to be explicit, some subtle issues are exposed that usually get glossed over. On the other hand, algebraic semiotics formalizations are grounded in social reality, contrary to much of classical semiotics. Dates and windows with scrollbars are discussed in detail to illustrate the approach.

Acknowledgements I thank Kai Lin for maintaining BOBJ, and Fox Harrell for some valuable comments. I also thank the students in my classes CSE 171 and 271 for their feedback.

2 Algebraic Semiotics

This section briefly reviews key concepts of algebraic semiotics and its application to interface design [9, 7, 10, 13, 8, 11], assuming some familiarity with the relevant mathematics. Algebraic semiotics incorporates major insights from the founders of semiotics, Charles Saunders Peirce [18] and Ferdinand Saussure [19]. Peirce emphasized (among other things) that the relation between a given token and its object is not just a function (as in denotational semantics), but a relation that depends on the situation in which the token is interpreted, while Saussure emphasized (among other things) that signs come in systems. Neither Peirce nor Saussure considered representations of sign systems, dynamic signs, the values of user communities, or pragmatic aspects of design.

Mathematically [9], a **semiotic system** or **semiotic theory** or **sign system** (we use these terms interchangeably) consists of a **signature** (which declares sorts, subsorts, operations, and

*This material is based on work supported by the National Science Foundation under Grant No. 9901002.

perhaps some fixed data sorts) with some axioms[1]; in addition, there is a level ordering on sorts (having a maximum or "top" element) and a priority ordering on the constructors at each level. Sorts classify the parts of signs, constructors build new signs from given parts, and data sorts classify the values of attributes of signs (e.g., color and size). Levels express the whole-part hierarchy of complex signs, whereas priorities express the relative importance of constructors and their arguments; social issues play an important role in determining these orderings.

There is a basic duality between theories and models: A semiotic theory determines the class of models that satisfy it, which we call its **semiotic space**; and a class of models has a unique (up to equivalence) most restrictive theory whose models include it; this duality is a Galois connection[2]. We may also use the term "semiotic space" for a semiotic theory, which, though potentially confusing, is justified by the duality. The main reason to prefer theories over models as a basis is that theories define spaces of signs, with axioms constraining the allowable signs, and hence the allowable representations; another is the ease of treating levels and priorities. For example, in formalizing the space of books, we want to allow anything with the right structure as a model; and in formalizing the representation that produces indices from books, we may well want to impose two axioms on the target space of indices, requiring that indexed items must be phrases of 1, 2 or 3 words, and that the page total for indexed phrases must be not more than 2% of a book's page total.

The following are some further informal examples of semiotic systems: dates; times; bibliographies (in one or more fixed format); tables of contents (e.g., for books, again in fixed formats); newspapers (e.g., the *New York Times* Arts Section); and a fixed website, such as the CNN homepage (in some particular instance of its gradually evolving format). Note that each of these has a large space of possible instances, but fixed structure. The next two sections explain two examples in detail. Section 3 discusses dates, using traditional initial model semantics (see [15] for an introduction to this, as well as to OBJ), while the second example, windows with scrollbars, discussed in Section 4, uses hidden algebra to model the dynamic aspect of these signs, since what is displayed in a window must change in concert with what its scrollbar shows. This paper is not the place for an exposition of hidden algebra (see [9, 16] for that), but we note that the elements of hidden sort in a model of a hidden theory are states, which can be changed by operations in the signature.

Mappings between structures became increasingly important in twentieth century mathematics and its applications; examples include linear transformations (and their representations as matrices), continuous maps of spaces, differentiable and analytic functions, group homomorphisms, and much more. Mappings between sign systems are only now appearing in semiotics, as uniform representations for signs in a source space by signs in a target space. Since we formalize sign systems as algebraic theories with additional structure, we should formalize **semiotic morphisms** as theory morphisms; however, these must be *partial*, because in general, not all of the sorts, constructors, etc. are preserved in real world examples. For example, the semiotic morphism that produces an outline from a book, omits the sorts and constructors for paragraphs, sentences, etc., while preserving those for chapters, sections, etc. In addition to the formal structure of algebraic theories, semiotic morphisms should also (partially) preserve the priorities and levels of the source space. The extent to which a morphism preserves the various features of semiotic theories determines its quality, as we will show in examples below.

Note that we take the direction of a semiotic morphism to be the direction that models or instances are mapped. Thus, if B is is a semiotic system for books and T one for tables of contents, then books (which are models of B) are mapped to their tables of contents, which are models of T. However, this map on models is determined by, and is dual to, the theory inclusion $T \to B$, which reflects the fact that the structure of tables of contents is a substructure of that of books. (This duality is also consistent with the fact that blending diagrams are drawn "upside down" in algebraic semiotics, compared with cognitive linguistics.) On the other hand, when

[1]This paper uses only algebraic signatures (with subsorts and data sorts), and as axioms, equations and sort constraints, though other logical systems could certainly be used instead. See [15] for a relatively easy exposition of algebraic signatures and equations; sort constraints are explained in Section 3.

[2]As in the theory of institutions [12], but note that this duality does not involve levels or priorities

initial model semantics is used, as in Section 3, the models map in the same direction as the theory morphism[3].

Note also that our use of the word "space" conflicts with that in cognitive linguistics, where **conceptual spaces** are actually single models, rather than classes of models. Moreover, conceptual spaces only have constants (called "elements") and relations among them; they do not have multi-argument constructors, sorts, levels, priorities, or axioms. For this reason, we should also distinguish conceptual blending, alias conceptual integration, from **structural blending**, which we may as well also call **structural integration**, where the former is blending of conceptual spaces and the latter is blending of semiotic systems, as theories, in general involving non-trivial constructors. For example, the integration of a window with its scrollbar is structural, not conceptual, although the conceptual aspects can also be studied (as in Section 6). Algebraic semiotics also goes beyond conceptual spaces to provide entities with states. These features are necessary for applications to user interface design, but also seem likely to have other applications[4].

Whereas conceptual spaces are good for studying meaning in natural language, they are not adequate for user interface design and other applications where *structure* is important, such as web design and music. For example, conceptual spaces and conceptual blending can help us understand *concepts about* music, but semiotic spaces and structural blending are needed for an adequate treatment of the *structure of* music, e.g., how a melody can be combined with a sequence of chords. Conceptual spaces are good for talking about concepts about (e.g., how we talk about) things, but are awkward for talking about the structure of things. Note also that greater cultural variation can be found in conceptual blending than in structural blending, because the former deals with concepts about something, whereas they latter deals with the structure of its instances and/or its representations.

There are at least three "modes" in which one might consider representations: *analytic*, *synthetic*, and *conceptual*. In the analytic mode, we are given one or more sign from the representation (i.e., the target) space, and we seek to reconstruct both the source space and the representation. In the synthetic mode, we are given the source space and seek to construct a good representation for the signs in that space, using some given technology (such as command line, or standard GUI widgets, or 3D color graphics) for the target space. In the conceptual mode, we seek to analyze the metaphorical structure of the representation, in the style of cognitive linguistics [21, 4]; for example, how is Windows XP like a desktop, or how is a scrollbar like a scroll? A treatment in this mode will involve conceptual spaces, in the sense of cognitive linguistics; see Section 6. In each mode, particularities of the cultures involved may be very significant, as we shall see.

There are at least two perspectives that one might take towards the study of signs and representations: *pragmatic* and *theoretical*. The first is the perspective of a designer, who has a job to get done, often within constraints that include cost, time, and stylistic guidelines; we may also call this an engineering perspective, and it will generally involve negotiating trade-offs among various values and constraints. The second is the perspective of a scientist who seeks to understand principles of design, and is thus engaged in a process of constructing and testing theories. From the second perspective, it makes sense to describe the semiotic spaces involved in a detailed formal way, and to test hypotheses with calculations and experiments with users. But from the pragmatic perspective, it makes sense to formalize only where this adds value to the design process, e.g., in especially tricky cases, and even then, only to formalize to the minimum extent that will get the job done. Experience shows that one can often get considerable benefit from applying principles of algebraic semiotics, such as identifying and preserving key features of the source space, without doing a great deal of formalization.

On the other hand, for designing safety critical systems, or for developing a deeper scientific understanding of design, one might wish not only to construct a formal mathematical model, but also to make it executable, so that it can be debugged, for example, using some version of

[3]Because these model spaces consist of only initial models, the map between them is the free functor, which is adjoint to, and in the opposite direction to, the forgetful functor induced by the theory morphism; hence it goes the *same* direction as the theory morphism, rather than the opposite direction. See [12] for more on these concepts.

[4]For example, some cognitive linguists have claimed that blending should be considered a basis for grammar; but this seems likely to require multi-argument constructors.

OBJ, such as OBJ3, CafeOBJ [3], Maude [2] or BOBJ [16][5]; all these are precise, fully formal mathematical notations, based on various forms of equational logic. If concision is more important than executability, then a strongly typed first order logic might be used instead.

From either the pragmatic or theoretical perspective, one should seek to model semiotic spaces as simply as possible, since this will simplify later tasks, whether they are engineering design or scientific theorizing and experimentation (not forgetting that the conceptual simplicity of a theory does not necessarily correspond to the simplicity of its expression in any particular language). A famous quotation from Einstein is relevant here:

> *Everything should be as simple as possible, but no simpler.*

However, from a pragmatic perspective, good representations need not be the simplest possible, for reasons that include engineering tradeoffs, the difficulty (and inherent ambiguity) of measuring simplicity, and social and cultural factors, e.g., relating to esthetics. Similar considerations apply, though to a notably lesser extent, to the simplicity of semiotic theories, since creating such theories is itself a design task, subject to various trade-offs. It may be reassuring to be reminded that in general there is no unique best representation.

3 Analysis of Dates

Let's begin with dates and their representations. Dates were also discussed in [9] and [7], but it is interesting, and not very difficult, to do the analysis with much greater precision; this example is also interesting because of the way that it highlights the possibility of different representations being best for different purposes, where these purposes depend on the particular situations of users, including their culture. This example does not require dynamic entities with state, though it could be reformulated to do so; some dynamic signs are considered in the next section.

We first introduce the data sorts that will be used in examples throughout this paper: the sort `Nat` of natural numbers, from the builtin module `NAT`; the sort `Id` of quoted identifiers, from the builtin module `QID`; and the sort `Float` of floating point numbers, from the builtin module `FLOAT`. Note that the sort `Bool` for the booleans, from the builtin module `BOOL`, is automatically imported by every module, without having to be mentioned.

A convenient source theory specifies dates as triples of natural numbers satisfying appropriate constraints[6]:

```
dth DATE is
  sort Date .
  pr 3TUPLE[NAT, NAT, NAT] *(sort Tuple3 to Date?,
       op (1*_) to (day_), op (2*_) to (month_), op (3*_) to (year_)).
  vars D M Y : Nat .  subsort Date < Date? .
  mb << D ; M ; Y >> : Date if 0 < D and D <= 30 and 0 < M and M <= 12 .
end
```

The keyword pair `dth...end` marks the opening and closing of this module, with `DATE` given as its name. The first line of its body declares a new sort named `Date`, while the second line imports the generic module `3TUPLE`, instantiates it with three instances of the data sort `Nat`, imported via the builtin module `NAT`; the main sort of this newly created module is then renamed from `Tuple3` to `Date?`, for things that might or might not turn out to be dates; and its three selectors are also renamed. Here `3TUPLE` is a builtin BOBJ generic module, having the constructor `<<_,_,_>>`, and the three selectors `1*_`, `2*_`, and `3*_`. The next line introduces three `Nat`-valued variables, and after that, a subsort `Date` of `Date?` is introduced; it will be used for those triples that satisfy the constraint. The final, most interesting, line gives the constraint on dates. The keyword `mb` indicates what in Maude is called a membership declaration, and although we adopt the `mb` notation of Maude, we prefer to call these **sort constraints**, since they define which elements

[5]"OBJ" is the family name, while "CafeOBJ," "OBJ3," "BOBJ," etc. name specific members of the family (although Maude is an exception to this convention).

[6]That there are 12 months, and that months have 30 days; in a more precise description, months would have different numbers of days, and there would be leap years, but we give a simplified version for expository purposes.

of a supersort must also belong to the subsort. The division into lines is arbitrary, because the BOBJ parser ignores carriage returns and linefeeds; hence we will often compress code by placing phrases on the same line.

It is clear that `Date` is the top level sort, and that `day`, `month`, and `year` are selectors for what could be seen as secondary sorts. However, because these have not been given explicit constructors, we cannot express a priority ordering on secondary constructors; although we could do so with a good deal of extra trouble, it is much simpler to enrich the theory with the new notion of **selector priority orderings**, which are optional partial orderings on the selectors of certain constructors[7]. It is well known that Europeans prefer to have the day come first, then the month, then the year, whereas Americans prefer to have the month first, then the day, then the year. These orderings cannot be directly expressed in BOBJ, and although they could be expressed indirectly, this would not do much good, so we just indicate them informally, with the following notation:

```
day >> month >> year
month >> day >> year
```

Although both orderings are in common use, the European one is more rational, because it preserves the natural ordering of these units by their increasing size. We can formalize this by looking at the coefficients in the formula for the difference (in days) between two dates,

```
<< D ; M ; Y >> - << D' ; M' ; Y' >> = (D - D') + 30(M - N') + 365(Y - Y').
```

This makes explicit by how much years are larger than months, and months than days.

This illustrates that the socially most preferred orderings are not necessarily the most rational, which can present a designer with a potentially confusing trade off. In this case, it is not difficult to see that the socially most preferred ordering should be used, and that if there is no definite social preference, then the more logical European ordering should be used.

But notice that, for some purposes, an equally reasonable ordering by *decreasing* size would be more useful than either ordering discussed above:

```
year >> month >> day
```

For example, a list of quarterly net earnings of some company with release dates of the quarterly reports, would be easier to scan if the year came first. However, in most everyday situations, the year is known, and the more rapidly changing item, which is the day, is the least likely to be known (although people may also be more likely to forget the month in certain situations). In fact, this ordering is used in China and some other countries. Notice also that this ordering is also consistent with the usual ordering for times, where

```
hour >> minute >> second
```

whereas the European ordering is not.

We will restrict consideration to representations of this semiotic space into a semiotic space of strings of characters, although there are certainly other very interesting representations, such as position in a calendar (and there are many kinds of calendar). There are four especially well known representations of dates as character strings; these can be classified by two binary variables, European vs. American, and numerical vs. mixed. The European representations put day first, then month, followed by year, while the American representations put month first and day second. The numerical representations separate the three components with `-` or `/`, while the two mixed representations write out the months as words.

To formalize this, we need to define a target space of strings of characters. For this, it is convenient to use a generic list module, which will be instantiated several times in this paper:

```
dth LIST[X :: TRIV] is sort List .
  pr NAT .  subsort Elt < List .
  op nil : -> List .
```

[7]This enrichment of the theory illustrates how the detailed study of concrete examples can contribute to scientific progress. This new feature actually adds no expressive power, but it can greatly simplify theories. Note that selector priority orderings should be considered part of the semiotic theory, like level orderings and constructor priority orderings, not part of the algebraic theory of the data structures involved.

```
  op __ : List List -> List [assoc].
  op |_| : List -> Nat .
  var X : Elt .  var L : List .
  eq |nil| = 0 .
  eq |X| = 1 .
  eq |X L| = 1 + |L| .
end
```

Here the phrase `[X :: TRIV]` defines the interface of the generic module; `X` is a formal parameter, and `TRIV` is a builtin interface theory, which says that any theory with a designated sort can be used as an actual parameter; it includes a formal sort parameter, designated `Elt`. The operation `__` is the constructor for lists, `nil` is the empty list, and the operation `|_|` gives the length of a list. The so-called "attribute" `[assoc]` of the `__` operation declares it to be associative.

Since BOBJ does not provide a character data type, but does provide both words and natural numbers, it is convenient to take lists of the union of these two sorts for our target space theory:

```
dth CHS is sort Ch .
  pr NAT + QID + LIST[Ch] *(sort List to ChList).
  subsorts Nat Id < Ch .
end
```

Elements of the builtin module `QID` have the sort `Id` and have forms such as `'A`, `'abc`, `'a21`, etc., so that elements of the sort `ChList` include things like `9 'May 2003`. To get representations of the form `5 / 9 / 2003`, we can just rename the list constructor, as follows:

```
dth CHS/ is pr CHS *(op (__) to (_/_)).  end
```

It remains to define the representation morphisms (see also the discussion of "co-relations" at the end of Section 3). This can also be done very easily in BOBJ, by defining a function from the top sort of the source theory to the top sort of the target theory; we can even use the notation of denotational semantics.

```
dth DATE-REP is pr DATE + CHS .
  op E[[_]] : Date -> ChList .
  op A[[_]] : Date -> ChList .
  var D : Date .
  eq E[[ D ]] = day(D) month(D) year(D) .
  eq A[[ D ]] = month(D) day(D) year(D) .
end
```

(We do not have to import any modules for data sorts, because these are automatically imported via the modules `DATE` and `CHS`, which have already imported them.) These are straightforward semiotic morphisms, each of which preserves one of the two most common orderings on the selectors of the date constructor. It is easy to test this morphism in BOBJ, as follows:

```
red E[[ << 9 ; 5 ; 03 >> ]] .
red A[[ << 9 ; 5 ; 03 >> ]] .
```

These two commands produce the following output (it is also interesting to notice the form of the date in this output), from which some material indicating successful processing of the various modules has been deleted:

```
                    \||||||||||||||||/
                    --- Welcome to BOBJ ---
                    /||||||||||||||||\
      BOBJ version 0.9.220 built: Sun May 11 03:02:23 PDT 2003
             University of California, San Diego
                 Sun May 11 19:27:46 PDT 2003

==========================================
dth DATE-REP
```

```
==========================================
reduce in DATE-REP : E [[(<< 9 ; 5 ; 03 >>)]]
result List: 9 5 03
rewrite time: 59ms       parse time: 10ms
Warning: non-termination corrected
==========================================
reduce in DATE-REP : A [[(<< 9 ; 5 ; 03 >>)]]
result List: 5 9 03
rewrite time: 32ms       parse time: 11ms
Warning: non-termination corrected
==========================================
```

The "`Warning: non-termination corrected`" message arises because of the way that sort constraints are implemented in BOBJ; it is not a cause for concern. The representations using the character `/` are almost the same:

```
dth DATE-REP/ is pr DATE + CHS/ .
  op E[[_]] : Date -> ChList .
  op A[[_]] : Date -> ChList .
  var D : Date .
  eq E[[ D ]] = day(D) / month(D) / year(D) .
  eq A[[ D ]] = month(D) / day(D) / year(D) .
end
```

Of course, this representation can be (and has been) tested the same way. The mixed case can be done almost as easily; we must specify the mapping of month numbers to month names, but we do not need to rename the list constructor; this is left as an exercise.

An alert reader might wonder why we defined these representations in what seems like a denotational style, when our theory calls for semiotic morphisms, which are almost supported by the BOBJ `view` feature. Actually, we *do* use BOBJ views, but since they do not support partiality, we define a supertheory, called a **co-relation**, which specifies how entities in the two theories are connected, and into which both the source and target theory are included by injective views. This is motivated by the following:

1. Semiotic morphisms are not functions, because in general they are only partially defined on the items to be represented. Moreover, the theories of complex sign systems, such as user interfaces, typically employ auxiliary constructions that *should not* be mapped into the space of representations.
2. The abstract (category theoretic) notion of a relation between spaces A, B is a space R and two maps, $A \leftarrow R \rightarrow B$, which can be thought of as projecting "pairs" in R to their two components.
3. Under the duality discussed in Section 2, on the theory level a relation in the above sense becomes a theory C with two inclusions $A' \rightarrow C \leftarrow B'$, where A', B' are the theories of A, B, respectively, i.e., it is a co-relation between the theories.
4. Any co-relation induces a dual relation between the corresponding spaces of models.

Thus, co-relations are a convenient way to represent semiotic morphisms using the capabilities of BOBJ, in contrast to a denotational approach, which is strictly functional. There is an analogy here with Peirce's semiotic triangle, which is also relational, in contrast to Saussure's more functional view of signification.

Finally, we consider a "unified numerical" representation, which counts the total number of days that have passed since 1 January 0000, given by the following module:

```
dth DATE-NUM-REP is pr DATE .
  op N[[_]] : Date -> Nat .
  var D : Date .
  eq N[[ D ]] = day(D) + (30 * month(D)) + (365 * (2000 + year(D))) .
end
```

This assumes the European ordering of day, month and year, and also assumes the twenty-first century; the value of `N[[ << 9 ; 5 ; 03 >> ]]`, our standard example, is 731,254 (days). It is interesting to consider why it is such a bad representation. This is an instance of the important question of measuring the *quality* of representations. It is obvious from the American vs. European representations that social convention can play an important role. But there are also some important structural considerations, of which we mention here just three:

1. The most important subsigns of a sign should map to correspondingly important subsigns of its representation (more technically, this calls for preserving important sorts and constructors).
2. It is better to preserve form (i.e., structure) than content, if something is sacrificed[8].
3. The most important axioms about signs should also be satisfied by their representations.

These principles are a basis for comparing the date representations given above. Whereas the American and European representations preserve the structuring into day, month and year in the source space `DATE`, the numerical representation does not, thus violating the first principle; for example, there is no subsign for years. The American and European representations lose some content (because we need to know the century), while the numerical representation does not; however, the first two representations are still better because they preserve form, as predicted by Principle F/C. Finally, the axioms (which are sort constraints) in `DATA` are not preserved by the numerical representation, because there are no subsigns to which they can refer, but they are preserved by the other representations, although they may take a different form (e.g., if months are represented by names).

For a different example of the third quality principle, the source theory for time of day in minutes has an important axiom expressing its cyclic nature, $s^{1440}(t) = t$, where s is the unary next-minute or "click" function, and t is a variable for time. This axiom is elegantly satisfied by the familiar circular clock, because it satisfies the stronger axiom $s^{720}(t) = t$.

We now introduce a generalized lexicographic ordering that can be used in comparing the quality of representations. Suppose $\leq_i$ is a partial order on a set R for each $i \in I$, where I is a finite partially ordered set, with ordering $<$ and maximum element $\top$. Let $r \equiv_i r'$ mean $r \leq_i r'$ and $r' \leq_i r$, let $r <_i r'$ mean $r \leq_i r'$ and $r' \not\equiv_i r$, and let $r \perp_i r'$ mean $r \not\leq_i r'$ and $r' \not\leq_i r$. Let $i \prec j$ mean that $i < j$ in I and there is no $k \in I$ such that $i < k < j$ in I. We now define the lexicographic product $\ll = \bigotimes_{i \in I} <_i$ on R, of the $<_i$ over I, by first defining orderings $\ll^i$ on R for $i \in I$, as follows: $r \ll^i r'$ iff $r <_i r'$, or else ($r \equiv_i r'$ or $r \perp_i r'$) and ($r \ll^j r'$ or $r \perp^j r'$) for all $j \prec i$, and $r \ll^j r'$ for at least one $j \prec i$. Then $r \ll^i r'$ iff $r <_i r'$ when i is minimal in I. Finally, we let $\ll$ be $\ll^\top$.

We may apply this as follows: Let R be a set of representations of some semiotic theory T, let $<_i$ be partial quality orderings on R, and let I with $<$ reflect the relative importance of these orderings. For another example, C might be the constructors of T, with L the levels of T, with $<$ the level ordering, and with $<_\ell$ the priority ordering on things of level $\ell \in L$. Then $\bigotimes_{\ell \in L} <_\ell$ on R combines the level and priority orderings in the correct way. If we now let R be some representations of T, let C with $<$ be as above, and let $r <_c r'$ iff r preserves c better than r' does[9], then $\ll$ is a useful quality measure for representations of T (though not the only one, since it does not take account of axioms). For example, the American and European representations are each better with respect to their own priority ordering, and both are better than the numerical representation, with either priority ordering. Further discussion of quality measures for morphisms is given in [15, 11], where it is explained how they relate to Peirce's classification of signs as symbolic, indexical, and iconic [18], and where it is noted that in general, each application requires its own carefully crafted "designer ordering"; the lexicographic construction described above is intended as a technical tool to ease the definition of such orderings.

[8] This is called **Principle F/C** [15]; although special cases are familiar to designers in many specialized areas, e.g., see [20], this may be the first general statement of the principle.

[9] For example, r' preserves every argument place of c that r does, and possibly more; or more generally, $<_c$ is the lexicographic product of the argument place preservation relations of c over their priority ordering.

4 Analysis of Scrollbar and Window

Our second example is windows with scrollbars, as in GUI operating systems. Scrollbars differ from dates in having states, so that initial algebra semantics is awkward; we therefore use hidden algebra semantics (e.g., [16]). This section considers scrollbars, and the next blends them with windows. We start with the source theory of "pointed files," which are text files with a pointer to the top line of the displayed text. We first specify data theories for lines and files, noting that the `dfn` feature in the second line of the body below abbreviates

```
pr LIST[QID] *(sort Id to LongLine).
```

and that `psort` gives a way to make a sort other than the first one introduced into the principal sort of a module, so that it can be the default choice when instantiating parameterized modules. Files are a data theory, while pointed files are a *behavioral* theory, with its "hidden" sorts representing states.

```
dth FILE is sort Line .
  dfn LongLine is LIST[QID] .
  subsort Id < Line < LongLine .
  var L : LongLine .
  mb L : Line if |L| <= 78 .
  pr LIST[Line] *(sort List to File, op (__).File to (_._)).
  psort File .
end

bth PTR-FILE is pr (NAT || FILE) *(sort Tuple to PtrFile?) .
  sort PtrFile .  subsort PtrFile < PtrFile? .
  var P : Nat .  var F : File .
  mb < P, F > : PtrFile if P <= |F| .
end
```

The keyword "`bth`" in the module `PTR-FILE` indicates that it defines states, which here have the hidden sort `PtrFile?`. The infix operation `||` is a builtin parameterized module that forms modules with a new (hidden) sort for states composed of the principal sorts of its two argument modules, with the constructor `<_,_>` of sort `Tuple`, and with selectors `1*` and `2*`, the same as for the parameterized module `TUPLE`. The sort constraints say that lines have 78 or fewer characters, and that proper pointed files have pointer value not greater than the length of the file.

Now we consider the target semiotic space for scrollbars. It is similar to the space for dates: scrollbars will have a `height` and two pointers, indicating the top and bottom of the highlighted part of the scrollbar, which we denote `top` and `bottom`, respectively. Height is actually a parameter which needs to be instantiated when a scrollbar is created, whereas top and bottom are parts of the state that vary as the scrollbar is used. Consequently, we need a parameterized theory for scrollbars, and therefore we first need a parameter theory to define its interface:

```
th HEIGHT is pr FLOAT . op height : -> Float .  end

bth SCROLLBAR[H :: HEIGHT] is sort Scrollbar .
  pr (FLOAT || FLOAT) *(sort Tuple to Scrollbar?,
       op (1*_) to (top_), op (2*_) to (bottom_)).
  subsort Scrollbar < Scrollbar? .
  vars T B : Float .
  mb < T, B > : Scrollbar if 0 <= B and B <= T and T <= height .
end
```

We define the relationship between `PTR-FILE` and `SCROLLBAR[H]` with the following co-relation:

```
bth PTR-FILE-REP[H :: HEIGHT] is
  pr (PTR-FILE || SCROLLBAR[H]) *(sort Tuple to PtrFileScrollbar).
  op [[_]] : PtrFile? -> Scrollbar? .
  var Pf : PtrFile? .
```

```
    eq [[ Pf ]]  = < (max(1* Pf, 40) * height)/ |2* Pf| ,
                     (max(0, 1* Pf - 40) * height)/ |2* Pf| > .
    var Pfs : PtrFileScrollbar .
    eq top    2* Pfs = top    [[ 1* Pfs ]] .
    eq bottom 2* Pfs = bottom [[ 1* Pfs ]] .
  end
```

This scrollbar representation of a subfile of a file, satisfies the three quality principles at the end of Section 4. It sacrifices content but preserves the file-subfile ratio. We can test this semiotic morphism by instantiating the generic scrollbar with a particular height, say 6 (inches), as given in the module H6 below, and then also making some assumptions about the pointer and file.

```
  dth H6 is pr FLOAT .  let height = 6.0 .  end

  bth PTR-FILE-REP6 is pr PTR-FILE-REP[H6] .
    let p = 250 .
    op f : -> File .
    eq |f| = 400 .
  end
  red  [[ < p, f > ]] .
```

The BOBJ output for this reduction is as follows:

```
  ==========================================
  reduce in PTR-FILE-REP6 : [[(< p , f >)]]
  result Scrollbar: < 3.75 , 3.15 >
  rewrite time: 1665ms       parse time: 20ms
  Warning: non-termination corrected
  ==========================================
```

This says that the top pointer is 3.75 from the bottom of the scrollbar, while the bottom pointer is 3.15 inches from the bottom. (The difference between these is .6 inches, one tenth of the scrollbar length, as it should be in this case, because 40 is 10% of 400.)

Working on this example forced consideration of the slightly complex way that scrollbars function near the bottom of files; although this is hardly a great mystery, it is something that I had never explicitly thought about before. Also, some new bugs were uncovered and corrected in the BOBJ system, and (as discussed before) the notion of selector priority was discovered. Phenomena like this are a typical and important part of scientific research, though they are not often reported.

5 Blending Scrollbar and Window

This section extends the analysis of the previous section to include the window as well as its scrollbar, and the link between them; indeed, what we normally call a "window" generally consists of both a scrollbar and an area for displaying something else, such as a text file. Such a system is a *structural blend* of its two components. The goal of this section is mainly to illustrate this notion of blend, rather than to explain the particular example. It is easy to define windows in the same style as the previous section, and then combine them with scrollbars:

```
  bth WINDOW is sort Window .
    pr FILE .  subsort Window < File .
    var F : File .
    mb  F : Window if |F| <= 40 .
  end

  bth WINDOW-SCROLLBAR is pr WINDOW || SCROLLBAR .  end
```

Thus, windows are objects with a state that is a file of not more than 40 lines, and a window-scrollbar is an object with a state having two components, a window and a scrollbar. However, this combination does not link what is in the window with what the scrollbar displays; for that, we need a more complex setup.

In cognitive linguistics (e.g., [21, 4]), blends have a so-called **generic space**, containing abstractions of concepts that occur in both input spaces, instances of which are not necessarily identified in the blended space. But in algebraic semiotics the material in this theory is shared, i.e., identified in the blend of the theories of the two interfaces. This is consistent with the mathematical foundations and methodology of [9, 7], but not with the intuitions in the cognitive linguistics literature. For this reason, the term **base theory** is used in [9, 7], rather than generic space. Our example needs the subfile representation in the scrolbar to correspond to the material displayed in the window. One possibility is a theory for a file with a subfile, which we might like to define as follows:

```
bth FILE-SUBFILE is pr (FILE || FILE) *(sort Tuple to FsubF?).
  sort FsubF .  subsort FsubF < FsubF? .
  vars F F' F1 F2 : File .
  mb < F , F' > : FsubF if F == F1 F' F2 .
end
```

which says that `F'` is a subfile of `F`; however, the implicit existential quantifiers over `F1` and `F2` in the condition cannot be handled by BOBJ, so we replace them by explicit Skolem functions:

```
bth FILE-SUBFILE is pr (FILE || FILE) *(sort Tuple to FsubF?).
  sort FsubF .  subsort FsubF < FsubF? .
  vars F F' : File .
  ops sk1 sk2 : File File -> File .
  mb < F , F' > : FsubF if F == (sk1(F, F') . F' . sk2(F, F')).
end
```

Next, we give two semiotic morphisms to knit window and scrollbar together into a blend. The first is the obvious inclusion view of `WINDOW` as a subtheory of `FILE-SUBFILE`, while the second is a co-relation between `FILE-SUBFILE` and `PTR-FILE`:

```
view WINDOW-TO-FILE-SUBFILE from WINDOW to FILE-SUBFILE is
end

bth C1 is pr (FILE-SUBFILE || PTR-FILE) *(sort Tuple to FsubFPtrFile).
  op [[_]] : FsubF -> PtrFile .
  vars F F' : File .
  eq [[ < F , F' > ]] = < |sk1(F,F')| + |F'| , F > .
  var Fspf : FsubFPtrFile .
  eq 1* 2* Fspf = |sk1(1* 1* Fspf, 2* 1* Fspf)| + |2* 1* Fspf| .
  eq 2* 2* Fspf = 1* 1* Fspf .
end
```

If to these we just add the obvious inclusion view of `WINDOW` into `WINDOW-SCROLLBAR` and the co-relation `PTR-FILE-REP6`, then the resulting diagram gives the blend we want.

5.1 Variable Size Window and Scrollbar

Variable size windows and scrollbars require a more elaborate construction. The following is one possible base theory, providing a file (in the sense of the module `PTR-FILE`) and two pointers, `T` and `B`, to lines in the file, for the top and bottom of the displayed material. The sort constraint just says that `B` must not be more than `T`, which must not be more than the length of the file:

```
bth FILE2PTR is
  pr PTR-FILE + (FILE || NAT || NAT) *(sort Tuple to File2Ptr?).
  sort File2Ptr .  subsort File2Ptr < File2Ptr? .
  vars T B : Nat .  var F : File .
  mb < F , T , B > : File2Ptr if B <= T and T <= |F| .
  psort File2Ptr? .
end
```

A more abstract theory that could be used instead of 3PTR, uses the length of the file and pointers to two locations within it, without giving the file itself.

```
bth 3PTR is pr (NAT || NAT || NAT) *(sort Tuple to 3Ptr?).
  sort 3Ptr .  subsort 3Ptr < 3Ptr? .
  vars A B L : Nat .
  mb < A , B , L > : 3Ptr if A <= B and B <= L .
end
```

That FILE2PTR is more concrete than 3PTR is expressed by the co-relation FILE2PTR+ below, which is FILE2PTR enriched with the more abstract representation:

```
bth FILE2PTR+ is pr FILE2PTR .
  pr (NAT || NAT || NAT) *(sort Tuple to  3Nat) .
  op |_| : File2Ptr? -> 3Nat .
  vars T B : Nat .  var F : File .
  eq |< F, T, B >| = < B, T, |F| >.
end
```

Then the view can be described quite simply as follows:

```
view V1 from 3PTR to FILE2PTR+ is sort 3Ptr? to 3Nat . end
```

Next, we give semiotic morphisms for building the blend. The co-relation C2 connects FILE-SUBFILE with FILE2PTR, and V3 is a default view, for which BOBJ can fill in all the mappings automatically, while the third instantiates the parameterized co-relation PTR-FILE-REP for a six inch scrollbar.

```
bth C2 is pr (FILE-SUBFILE || FILE2PTR) *(sort Tuple to FsubFF2Ptr).
  op [[_]] : FsubF -> File2Ptr .
  vars F F' : File .
  eq [[ < F , F' > ]] = < F , |sk1(F,F')| + |F'| , |sk1(F,F')| > .
  var S : FsubFF2Ptr .
  eq 1* 2* S = 1* 1* S .
  eq 2* 2* S = |sk1(1* 1* S, 2* 1* S)| + |2* 1* S| .
  eq 3* 2* S = |sk1(1* 1* S, 2* 1* S)| .
end

view V3 from PTR-FILE to FILE2PTR is
  sort PtrFile? to PtrFile? .
end

bth PTR-FILE-REP-VAR is pr PTR-FILE-REP[HEIGHT] .
  op winsize : -> Float .
  var F : File .
  eq height = (6 / 40) * winsize .
end
```

Putting this collection of theories and morphisms together gives the blend we want, in which the size of the window is given by the (user-definable) constant winsize.

5.2 What is Blending?

This subsection briefly discusses some points about structural blending and its relation to other traditions. We first note that two kinds of dynamics are involved in blending: the process of blending itself, and entities with internal states. Whereas cognitive linguistics has so far focussed mainly on the former, algebraic semiotics is more concerned with the latter, due to its focus on user interface design and similar applications. Our second point is that in structural blending, cross-space mappings emerge through the identifications implied by the base space; relations like causality are represented as ordinary relations (Bool-valued functions in BOBJ) rather than having

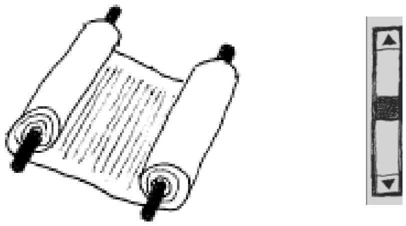

Figure 1: A Scroll and a Scrollbar

a special *ad hoc* status. Thirdly, the distinction between single and double scope blending seems a bit artificial in algebraic semiotics, because its applications typically involve multiple "scopes" arising from multiple spaces and morphisms among them. Fourthly, we are developing a formal characterization of blending, based on the notion of colimit [9], taking account of the quality orderings on morphisms, and allowing there to be more than one blend for a given system of spaces and morphisms. Finally, in contrast with much of classical semiotics, but in agreement with cognitive science in general, and cognitive linguistics in particular, we eschew belief in the Platonic reality of signs, sign systems, etc., even though there is a strong tendency in our culture to identify mathematical formalizations with such ideal entities.

6 Conceptual Analysis of Window and Scrollbar

This section sketches a conceptual analysis of scrollbars, mainly intended to show how this level of analysis differs from that of the previous sections, in that it focuses on questions like "How is a scrollbar like (and unlike) a scroll?" rather than questions like "How does a scrollbar represent window-based file display?" or "How good is that representation?" A major motivation for analyses at the conceptual level is to explore the cognitive consistency of designs. The appropriate tools for answering such questions are models of *how we think about* scrollbars, rather than models of scrollbars themselves. As in traditional cognitive linguistics, there is little payoff from being highly precise in such discussions, and we shall therefore be relatively informal.

The conceptual space for a scroll (see Figure 1) contains two rollers, a roll of parchment (or other material) with text on it, and the affordance[10] for scrolling through the text by rotating one or both of the rollers. Note that it is possible to expose any contiguous segment of the text with appropriate roller rotations.

A scrollbar is a bar with a highlighted sub-bar (in Figure 1, the sub-bar is highlighted by darkening); it represents an arbitrary contiguous segment of text, by the proportions of the locations of the boundaries of the highlighted sub-bar to the total length of the bar, as described in the module `PTR-FILE-REP-VAR`. Typically, affordances for moving the segment up or down are presented as arrowheads at the top and bottom of the bar, as in Figure 1, and there are also non-perceived affordances for making larger jumps up or down the text, by clicking on the corresponding non-highlighted areas (however many users are not aware of these affordances).

The analogy between the displayed text of a scroll and the highlighted sub-bar of a scrollbar is good, except that it is more convenient to use a real scroll with its rollers to the left and right of the text[11], whereas the scrollbars that represent displayed text are oriented vertically, not horizontally. This situation is reflected in the mathematics, which expresses the proportional representation quite nicely while ignoring orientation (see Section 5.1).

On the other hand, analogies for the affordances are not so close. First, scrolling the text by using the mouse to move the highlighted portion of the scrollbar is very different from any

[10] We understand affordances in the sense of Gibson, as potential interactions between an organism (here, the user) and the environment (here, the system) [5, 6] and "perceived affordances" in the sense of Norman [17] as affordances that the user is able to perceive as such, by virtue of what is displayed.

[11] This is because of the location of human hands on the left and right of the body. Moreover, in many old languages, the text is oriented vertically and read from top to bottom.

affordance of a real scroll; this indirect control arises through the structural blending of the window with its scrollbar. Operating one roller does not cause exposure of a different text of the same size, but rather increases or decreases the displayed text, from the left or right, depending on which roller is rotated in which direction. The behavior of the scrollbar controls is closer to that of the “up” and “down” buttons on an elevator than to the rollers on a scroll, though this behavior also differs from that of elevator buttons. The non-perceived scrollbar affordances do not correspond to any affordances of real scrolls.

7 Conclusions

The approach advocated in this paper is not limited to user interface design in the narrow sense, as the representations for dates and the conceptual analysis of windows demonstrate. One way to explore the potential for extended applications of algebraic semiotics is to think of “interfaces” in a very broad sense that includes any form of communication in any medium, which is already the scope of Piercian semiotics. For example, Principle F/C applies to the generation of an outline from a book, as well as to the navigational guides provided within buildings, such as floor numbers, room numbers, and lists of office occupants with their room numbers.

New ideas in this paper include formalization of dynamic signs using hidden algebra, orderings on the selectors of a given constructor, co-relations for blending, and generalized lexicographic order to measure representation quality, based on the importance of components. In addition, the following are some subtle points revealed during the process of formalization:

1. There is a rich duality between models and theories.
2. The maps induced on models by theory maps go in the opposite direction.
3. While initial semantics is good for static signs, hidden semantics (or some similar formalism) is needed for dynamic signs, in order to handle states.
4. Non-trivial constructors are needed for many examples, which therefore require structural blending instead of conceptual blending.

On the other hand, and in opposition to efforts at formalizing meaning such as situation semantics [1], algebraic semiotics stresses the necessity for grounding in social process, both before and after formalization, the first to obtain reasonable levels and priorities, and the second to check that the formalization is useful.

It is hoped that the examples in this paper will convince readers that algebraic semiotics provides a precise and useful tool for analyzing representations and interfaces, including not just specification techniques, but also some very general quality principles to aid design. However, it should be emphasized that in design practice, algebraic semiotics should generally be applied far more informally than the BOBJ code here might suggest. Practical application is still at an early stage, and only one serious case study has been done, the Tatami system [14, 15], but this promises to be an exciting area for future exploration.

References

[1] Barwise, J. and Perry, J. (1983). *Situations and Attitudes.* MIT (Bradford).

[2] Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., and Quesada, J. F. (2001). Maude: Specification and programming in rewriting logic. *Theoretical Computer Science.*

[3] Diaconescu, Răzvan. and Futatsugi, K. (1998). *CafeOBJ Report: The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification.* World Scientific. AMAST Series in Computing, Volume 6.

[4] Fauconnier, G. and Turner, M. (2002). *The Way We Think.* Basic.

[5] Gibson, J. (1977). The theory of affordances. In Shaw, R. and Bransford, J., editors, *Perceiving, Acting and Knowing: Toward an Ecological Psychology.* Erlbaum.

[6] Gibson, J. (1979). *An Ecological Approach to Visual Perception.* Houghton Mifflin.

[7] Goguen, J. (1996). Semiotic morphisms. Available on the web at `www.cs.ucsd.edu/users/goguen/papers/smm.html`. Early version in *Proc., Conf. Intelligent Systems: A Semiotic Perspective, Vol. II*, ed. J. Albus, A. Meystel and R. Quintero, Nat. Inst. Science & Technology, (Gaithersberg MD, 20–23 October 1996) pages 26–31.

[8] Goguen, J. (1996–2001). The UCSD Semiotic Zoo. Website at URL `www.cs.ucsd.edu/users/goguen/zoo/`.

[9] Goguen, J. (1999a). An introduction to algebraic semiotics, with applications to user interface design. In Nehaniv, C., editor, *Computation for Metaphors, Analogy and Agents*, pages 242–291. Springer. Lecture Notes in Artificial Intelligence, Volume 1562.

[10] Goguen, J. (1999b). Social and semiotic analyses for theorem prover user interface design. *Formal Aspects of Computing*, 11:272–301. Special issue on user interfaces for theorem provers.

[11] Goguen, J. (Spring 2003). User interface design class notes. The CSE 271 website, at `www.cs.ucsd.edu/users/goguen/courses/271/`.

[12] Goguen, J. and Burstall, R. (1992). Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39(1):95–146.

[13] Goguen, J. and Harrell, F. (2003). Information visualization and semiotic morphisms. In Malcolm, G., editor, *Visual Representations and Interpretations.* Elsevier. Proceedings of a workshop held in Liverpool, UK.

[14] Goguen, J., Lin, K., Roşu, G., Mori, A., and Warinschi, B. (2000). An overview of the Tatami project. In *Cafe: An Industrial-Strength Algebraic Formal Method*, pages 61–78. Elsevier.

[15] Goguen, J. and Malcolm, G. (1996). *Algebraic Semantics of Imperative Programs.* MIT.

[16] Goguen, J., Roşu, G., and Lin, K. (to appear 2003). Conditional circular coinductive rewriting. In *Recent Trends in Algebraic Development Techniques, 16th International Workshop, WADT'02.* Springer, Lecture Notes in Computer Science. Selected papers from a workshop held in Frauenchiemsee, Germany, 24–27 October 2002.

[17] Norman, D. A. (1988). *The Design of Everyday Things.* Doubleday.

[18] Peirce, C. S. (1965). *Collected Papers.* Harvard. In 6 volumes; see especially Volume 2: Elements of Logic.

[19] Saussure, F. (1976). *Course in General Linguistics.* Duckworth. Translated by Roy Harris.

[20] Tufte, E. (1983). *The Visual Display of Quantitative Information.* Graphics Press.

[21] Turner, M. (1997). *The Literary Mind.* Oxford.