ON VISUAL FORMALISMS

The higraph, a general kind of diagramming object, forms a visual formalism of topological nature. Higraphs are suited for a wide array of applications to databases, knowledge representation, and, most notably, the behavioral specification of complex concurrent systems using the higraph-based language of statecharts.

DAVID HAREL

Visualizing information, especially information of complex and intricate nature, has for many years been the subject of considerable work by many people. The information that interests us here is nonquantitative, but rather, of a structural, set-theoretical, and relational nature. This should be contrasted with the kinds of quantitative information discussed at length in [43] and [46]. Consequently, we shall be interested in diagrammatic paradigms that are essentially topological in nature, not geometric, terming them *topovisual* in the sequel.

Two of the best known topo-visual formalisms have their roots in the work of the famous Swiss mathematician Leonhard Euler (1707–1783). The first, of course, is the formalism of graphs, and the second is the notion of *Euler circles*, which later evolved into *Venn diagrams*. Graphs are implicit in Euler's celebrated 1736 paper, in which he solved the problem of the bridges of Königsberg [12]. (An English translation appears in [3].) Euler circles first appear in letters written by Euler in the early 1760s [13], and were modified to improve their ability to represent logical propositions by John Venn in 1880 [48, 49]. (See [19, chap. 2] for more information.[1])

A graph, in its most basic form, is simply a set of points, or nodes, connected by edges or arcs. Its role is to represent a (single) set of elements S and some binary relation R on them. The precise meaning of the relation R is part of the application and has little to do with the mathematical properties of the graph itself. Certain restrictions on the relation R yield special classes of graphs that are of particular interest, such as ones that are connected, directed, acyclic, planar, or bipartite. There is no need to elaborate on the use of graphs in computer science—they are used extensively in virtually all branches of the field. The elements represented by the nodes in these applications range from the most concrete (e.g., physical gates in a circuit diagram) to the most abstract (e.g., complexity classes in a classification schema), and the edges have been used to represent almost any conceivable kind of relation, including ones of temporal, causal, functional, or epistemological nature. Obviously, graphs can be modified to support a number of different kinds of nodes and edges, representing different kinds of elements and relationships.

A somewhat less widely used extension of graphs is the formalism of *hypergraphs* (see, e.g., [1]), though these are also finding applications in computer science, mainly in database theory (see [14], [15], and [31]). A hypergraph is a graph in which the relation being specified is not necessarily binary; in fact, it need not even be of fixed arity. Formally, an edge no longer connects a pair of nodes, but rather a subset thereof. This makes hypergraphs somewhat less amenable to visual representation, but various ways of overcoming this difficulty can be conceived (see Figure 1). In analogy with graphs, several special kinds of hypergraphs are of particular interest, such as directed or acyclic.

It is important to emphasize that the information

[1] Interestingly, both these topo-visual achievements of Euler were carried out during the period in which he could see with one eye only. (Euler lost sight in his right eye in 1735, and in the left around 1766.) It is tempting to attribute this in part to the fact that the lack of stereoscopic vision reduces one's ability to estimate size and distance, possibly causing a sharper awareness of topological features.

Part of this work was carried out while the author was at the Computer Science Department of Carnegie-Mellon University, Pittsburgh, Pennsylvania.

conveyed by a graph or a hypergraph is nonmetric and captured by the purely topological notion of *connectedness* (a term taken from [18]); shapes, locations, distances, and sizes, for example, have no significance.

Although not quite as widely used as graphs, Euler circles, or Venn diagrams, are often used to represent logical propositions, color charts, etc. (see Figure 2). The basic idea is to appeal to the two-dimensional case of the Jordan curve theorem (e.g., [11, 30]), which establishes that simple closed curves partition the plane into disjoint inside and outside regions. A set is then represented by the inside of such a curve,[2] giving the topological notions of *enclosure*, *exclusion*, and *intersection* of the curves their obvious set-theoretic meanings: being a subset of, being disjoint from, and having a nonempty intersection with, respectively.[3]

The bottom line is that, whereas graphs and hypergraphs are a nice way of representing a set of elements together with some special relation(s) on them, Euler/Venn diagrams are a nice way of representing a *collection* of sets, together with some *structural* (i.e., set-theoretical) relationships between them. The difference between the two types of relationships is obvious. The structural ones are uniformly interpreted in the obvious set-theoretic fashion, in much the same way as the = symbol in logical formalisms is uniformly interpreted as the equality predicate, whereas the edge relations of graphs and hypergraphs attain different meanings in different applications.

The main observation motivating the present work is that in numerous computer-related applications the complexity of the objects, systems, or situations under consideration is due in large part to the fact that *both* capabilities are needed. We have a (usually large) number of sets that are interrelated in nontrivial set-theoretic ways, but they are also related via one or more additional relationships of special nature, depending on the application at hand. Furthermore, among the structural, set-theoretic relationships it is often desirable to identify the *Cartesian product* of some of the sets—an action that can be crucial in preventing certain kinds of representations from growing exponentially in size. In line with these observations, which will be supported by examples in the sequel, the purpose of this article is to extend and combine Euler's two topo-visual formalisms into a tool suitable for dealing with such cases.

In the next section, we introduce *higraphs*,[4] first modifying Euler/Venn diagrams somewhat, then extending them to represent the Cartesian product, and finally connecting the resulting curves by edges or hyperedges. (The appendix contains the formal syntax and semantics of simple higraphs.) We will then illustrate the power of the formalism by briefly discussing higraph-based versions of such graphical languages as entity-relationship diagrams, semantic and associative networks, and dataflow diagrams. Later we will detail a less obvious application called *statecharts* [21], which are essentially a higraph-based version of finite-state machines and their transition diagrams.

HIGRAPHS

Let us start with a simple example of Euler circles (Figure 3). As can be seen, we prefer to use rounded rectangles, or rounded rectilinear shapes (*rountangles*?), rather than circles or unrestricted curves, and shall call the areas, or zones, they enclose *blobs* in the sequel. Second, as the formal definition supplied in the appendix shows, we regard each blob as denoting a certain kind

[2] Venn himself was not always consistent in this respect; see [49, p. 117] or [19, p. 43] for a description of his five-set diagram.

[3] The topological paradigm used here is termed *insideness* in [18].

[4] This is not a particularly successful choice of term, but was chosen nevertheless to be reminiscent of *high graphs* or *hierarchical graphs*, though our diagrams are not limited to being stratified in the way the word *hierarchical* might imply.

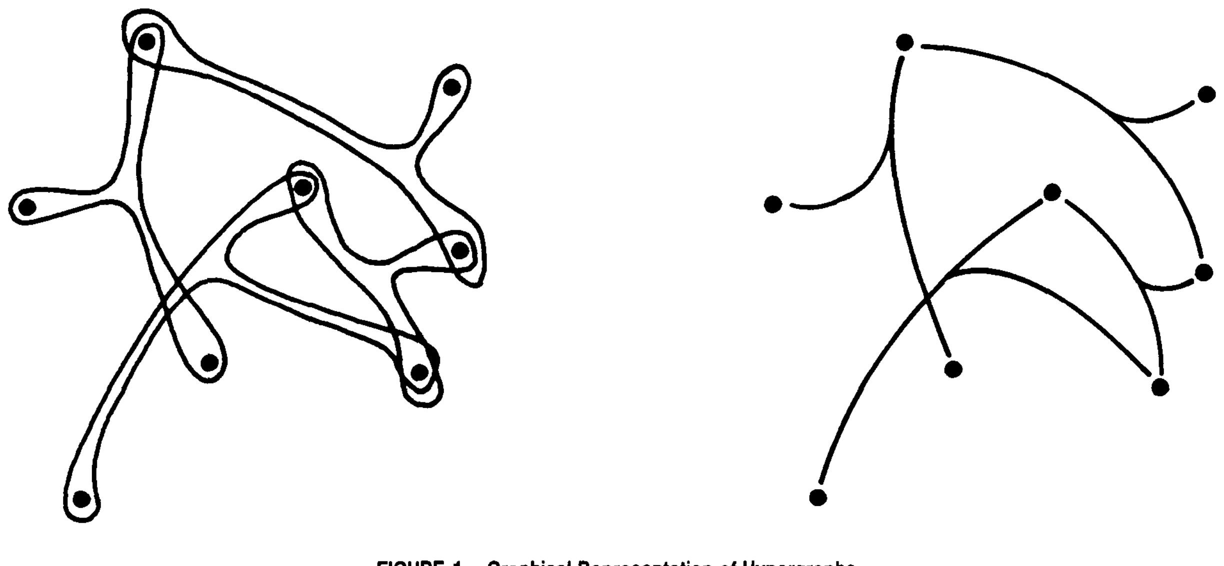


FIGURE 1. Graphical Representation of Hypergraphs

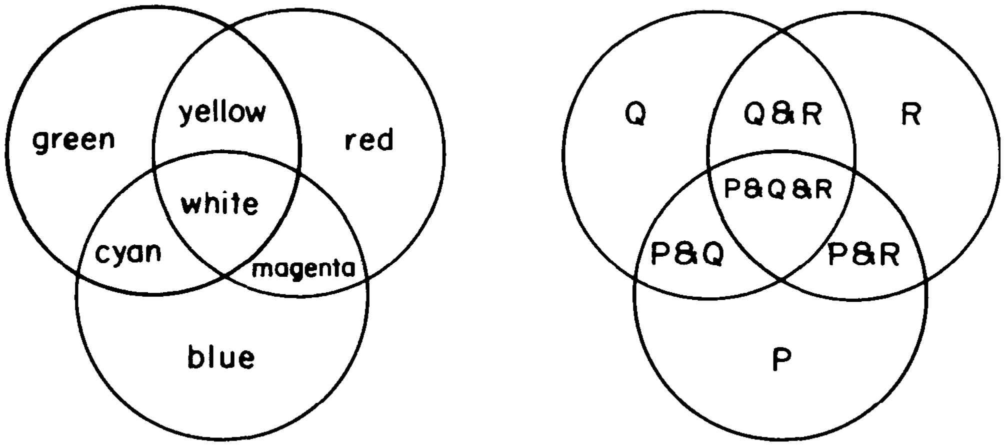


FIGURE 2. Applications of Euler Circles, or Venn Diagrams

of set, with the nesting of curves denoting set inclusion, not set membership. Thus, Figure 3 can be seen to contain several cases of inclusion, disjointness, and intersection of sets.

For our first real departure from Euler and Venn's treatment, we now require that every set of interest be represented by a unique blob, complete with its own full contour. One of the reasons for this is the desire to provide every set with its own area (e.g., for naming or labeling purposes). For example, does the *A* in Figure 3 represent the difference between the sets represented by the two large blobs, or the entire set on the upper left? The answer, following Venn's notational conventions, would appear to be the former; but then how do we label the upper set itself?

Our solution is illustrated in Figure 4, where the two large intersecting blobs are clearly labeled *A* and *D*, the intersection $A \cap D$ is labeled *C*, and the difference $A - D$ is called *B*. In fact, had we left out *B* and its contour we could not refer to $A - D$ at all. More precisely, with this "unique-contour" convention, the only real, identifiable sets are the *atomic* sets, that is, those represented by blobs residing on the bottom levels of the diagram, containing no wholly enclosed blobs within. Any other blob merely denotes the compound set consisting of the union of all sets represented by blobs that are totally enclosed within it. The atomic blobs of Figure 4 are thus *B*, *C*, *E*, *G*, *H*, *I*, *K*, *L*, *M*, *N*, *O*, *Q*, *S*, and, significantly, also *T*. The fact that *T*, as a Jordan curve, intersects *R* in Figure 4 does not necessarily mean that the sets represented by[5] *T* and *R* really intersect or that $T - R$ is nonempty. In fact, in our formalism, the intersection of two curves does not, in itself, mean anything since unless internal blobs appear in the appropriate places neither the difference nor the intersection of the sets they represent is itself identifiable. Thus, as far as the information present in Figure 4, *T* could just as well have been drawn completely dis-

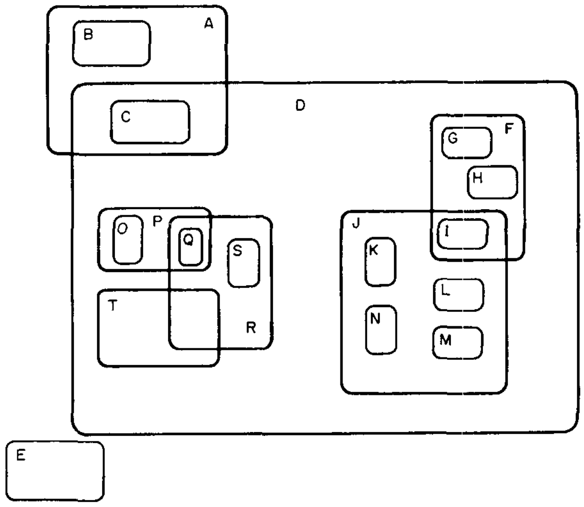


FIGURE 4. Adding Unique Contours for All Identifiable Sets

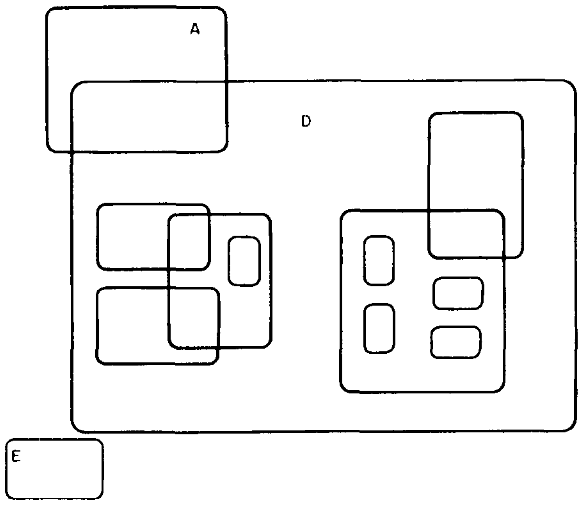


FIGURE 3. Simple Blobs

[5] In the sequel, we shall often blur the distinction between a curve, its associated blob, and the set it depicts.

joint from R, since R is defined by the figure to be the union of Q and S, whether T's curve intersects it or not. Of course, if T had been entirely enclosed within R, things would have been quite different, with R then being the union of Q, S, and T. All this might sound a little strange, but it is not really restrictive, since one can always let T and R intersect and simply add extra blobs representing $T \cap R$ and $T - R$, as is done in Figure 5.

Thus, one might say that empty space in our diagrams always represents nothing at all, except if it is the area of an atomic blob, which is one that contains no enclosed blobs. An atomic blob always represents some identifiable set, though clearly such a set might just happen to be an empty one.

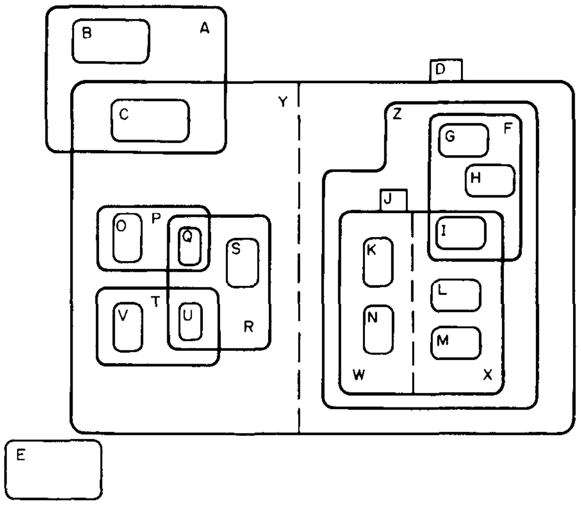


FIGURE 5. Adding Cartesian Products

We now add the ability to represent the *Cartesian product.* Figure 5 shows the notation used—a *partitioning* by dashed lines. In it J, for example, is no longer the union of K, N, I, L, and M, but, rather, the product of the union of the first two with the union of the last three. Symbolically,

$$J = W \times X = (K \cup N) \times (I \cup L \cup M).$$

We shall call the operands of the product, W and X in this case, the *orthogonal components* of blob J. Actually, the Cartesian product is *unordered*, in the sense that $A \times B$ is always the same as $B \times A$, so that J is really a set of unordered pairs of elements. Thus, our $\times$ operator is symmetric, and in fact, in the appendix we use the symbol $\otimes$, instead of $\times$, to denote it. Another consequence of this, and of our previous convention regarding set inclusion versus set membership, is that the product is also associative. In this way, if $c \in C$, $k \in K$, and $m \in M$, then the unordered triple $\{c, k, m\}$ would be a legal element of the set D of Figure 5, without the need to distinguish it from $\{c, \{k, m\}\}$. To make this idea work, it helps to assume that all atomic sets are pairwise disjoint (i.e., no element appears in any two of these sets).

Decomposing a blob into its orthogonal components by topologically partitioning the inner area of a Jordan curve yields a unique unambiguous area for each such component. Thus, the labels Y, W, and X in Figure 5 label the appropriate components unambiguously. On the other hand, as we shall see, there is another reason for wanting sets to have their own blob contours, and if so desired an orthogonal component can be enclosed in one of its own, as is Z in Figure 5. Notice the somewhat awkward location for the labels D and J. There are a couple of other possibilities for locating the label of a product blob, among which is the one illustrated in Figure 6, but we shall remain with that of Figure 5.

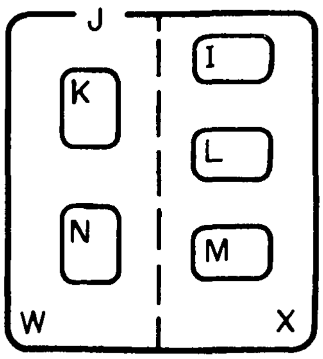


FIGURE 6. An Alternative for Labeling Partitioned Blobs

Now that we have a formalism for representing the sets we are interested in and their structural, set-theoretic relationships, it is time to add edges. A *higraph* is obtained by simply allowing edges, or more generally, hyperedges, to be attached to the contour of *any* blobs. As in graphs, edges can be directed or undirected, labeled or unlabeled, of one type or of several, etc. In Figure 7 we have allowed for a single kind of unlabeled directed hyperedge of arity between 2 and 3. Most of the arrows in the figure are simple binary edges, such

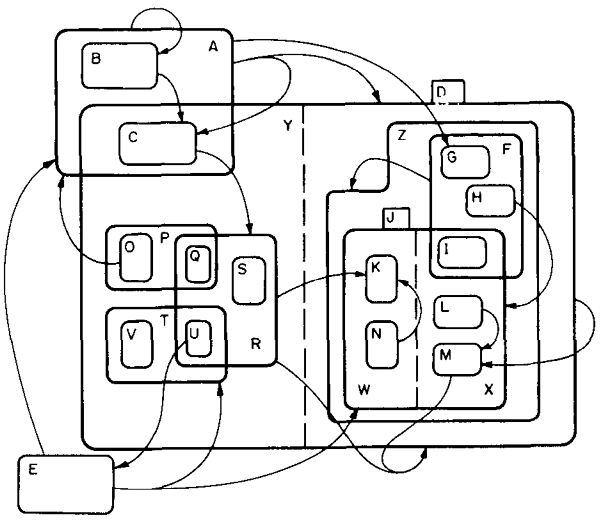


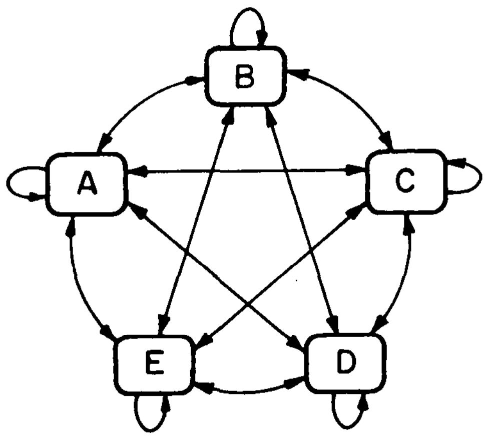
FIGURE 7. Adding Edges Resulting in a Higraph

as the very high-level one connecting E to A, the very low-level one connecting N to K, and the interlevel one connecting U to E. Others are directed three-way hyperedges, such as the one connecting E to both J and T, and the one connecting both R and M to D. Clearly, there is nothing to prohibit self-directed or partially self-directed edges, such as the one connecting A to its subblob B. The formal meaning of such edges (see the appendix) in the graph-theoretic spirit simply associates the target blobs with the source blobs via the particular relationship the edges represent. Here, then, is the other reason for wanting each set of interest to have its own contour: to enable it to be connected to others via the edges.

In the sequel the term *higraph* will be used in a very liberal sense, making no real distinction between the various possibilities, for example, the edge-based or hyperedge-based cases.

we are free to attach any meaning at all to the relationship itself and to the way (if any) that it extends downwards to the elements of those sets. Thus, if we take the relationship R represented by ordinary arrows in a higraph to mean "each element in the source set is related to *some* element in the target set by relationship T," then the information conveyed by Figure 9, for example, cannot really by captured by an ordinary graph with T-edges, since one would be forced to decide which element in the target set is meant, thus causing an overspecification.

The computer science literature is full of uses of graphs, and it appears that many of these can benefit from the extensions offered by higraphs. Consider the *entity-relationship* (*E-R*) *diagrams* used in the conceptual specification of databases [7]. These are really hypergraphs with a single type of node that is depicted by a rectangle and denotes an entity in the described pool of data. The hyperedges, whose labels are written in small diamond-shaped boxes (that should not be regarded as nodes), capture the intended relationships between entities. Figure 10 shows a simple example of such a diagram, representing a small part of the data used by an airline company.[6] Its information content is clear: `pilots` can fly `aircraft`, `secretaries` work for `employees`, and `employees` are paid `salaries` on certain `dates` (the latter being a three-way relationship). Notice, however, the `is-a` edges, informing us that `pilots` and `secretaries` are really `employees` too. These are conveying information of a totally different kind. Indeed, they capture precisely the kind of structural, set-theoretic relations discussed earlier. Using the very same "flat" diagrammatic representation for both kinds of relationships can cause a lot of confusion, especially in large and intricate cases, as a glance at some of the examples in the literature shows.



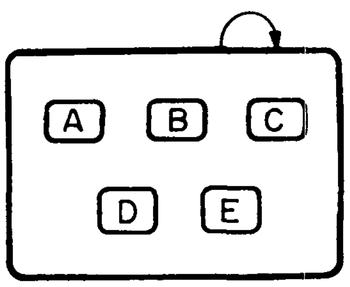


FIGURE 8. Two Representations of a 5-clique

SOME IMMEDIATE APPLICATIONS

The first thing to notice when attempting to apply higraphs is that edges connect sets to sets, not elements to elements as in graphs. The most common way of interpreting a higraph edge is as a collection of regular edges, connecting each element in one set with each element in the other. In this way, for example, it is possible to represent a 5-clique, as in Figure 8. This all-to-all semantics is not mandatory, however, since the bare meaning of a higraph edge is that the relationship it represents holds between the *sets* it connects. Hence,

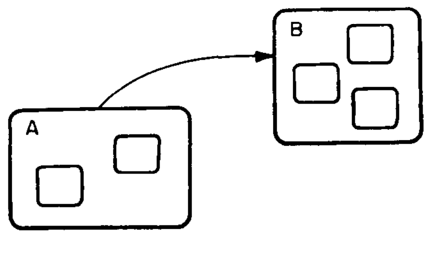


FIGURE 9. A Simple Higraph

[6] Actually, Figure 10 does some injustice to the E-R formalism, as it is sometimes called, by ignoring the additional features that the formalism supports, such as attributes for both entities, and relationships and the classification of relationships as one-one, many-one, etc. Throughout, we shall have to be satisfied with describing only those features of a formalism that are directly relevant to our discussion.

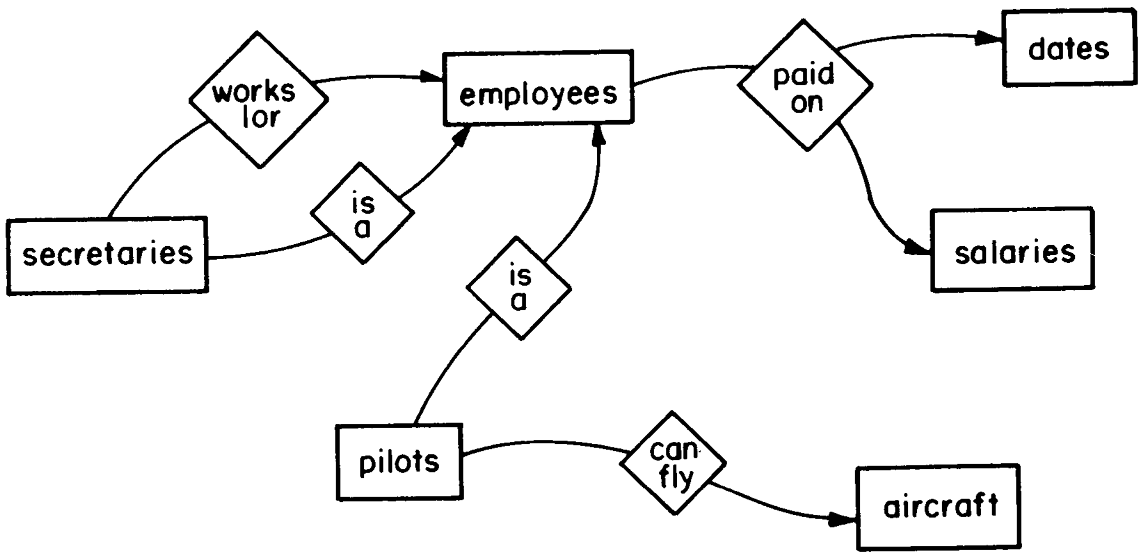


FIGURE 10. A Simple E-R Diagram

Figure 11 shows the way such information can be represented in a higraph-based extension of E-R diagrams. The set of employees is divided into the subsets of interest, secretaries and pilots (with an additional blob for all others, if so desired). The paid-on edge emanates from the employees blob, while the can-fly edge emanates from the pilots blob only—exactly what one would expect. The work-for edge rightly connects the secretaries blob with its parent blob—employees. The new information has been quite easily added: aircraft are now just part of the overall equipment, which is related to years by the relationship received-on, while the dates on which salaries are received have been specified as consisting of pairs from the orthogonal components month and year. Moreover, independent divisions can be represented by overlapping blobs, as illustrated in Figure 12, which shows how a new breakup of the employees by sex can be added to the previous figure

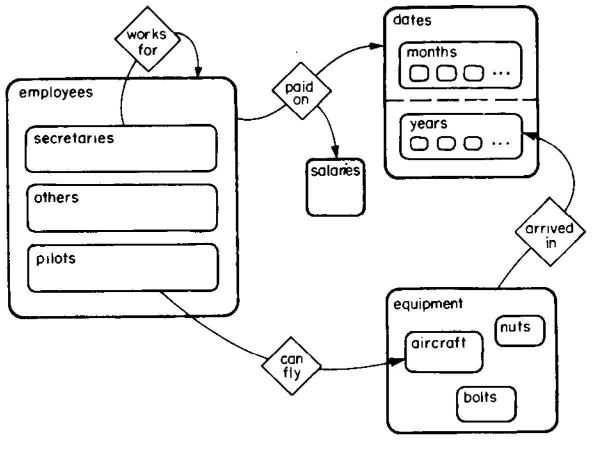


FIGURE 11. A Higraph-Based Version (and extension) of Figure 10

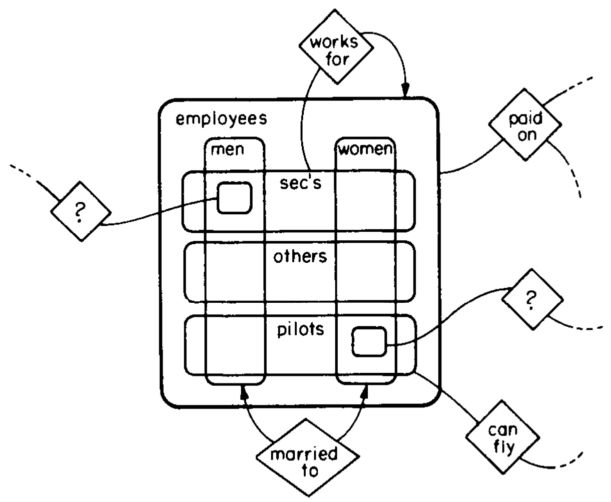


FIGURE 12. Two Breakups of Employees

with a couple of additional details. In it we might have reason to relate the female pilots or the male secretaries to other entities. In practice, overlaps should probably be used somewhat sparingly, as overly overlapping blobs might detract from the clarity of the total diagram, an observation that is in line with the often-made claim that a hierarchy is by far the way humans prefer to structure things (see [45, chap. 1]. This opinion is not universally accepted, however, so the human-factors aspects of formalisms like higraphs would appear to require careful experimental research, such as those carried out in [18] and [20].

Occasionally, authors have used other labels to capture is-a relationships, typically ones that try to describe the special nature of the breakup into subsets. As

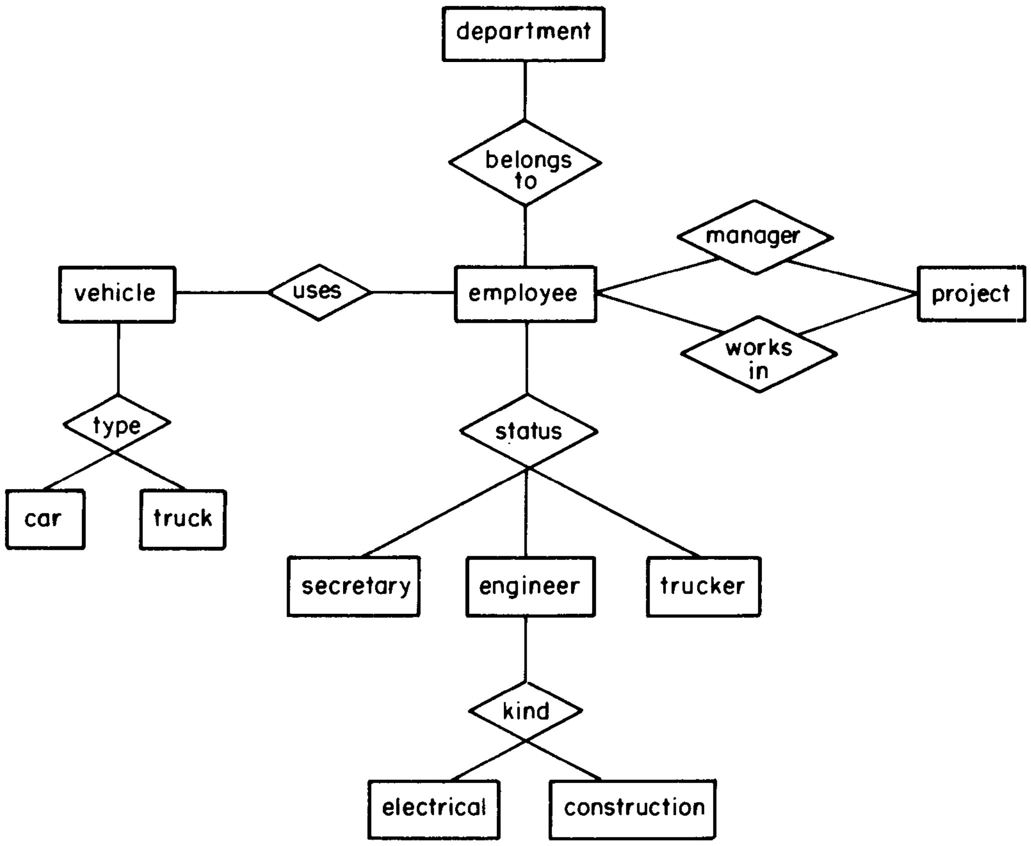


FIGURE 13. Another E-R Diagram (taken from [39])

an example consider Figure 13, which is Figure 9 of [42] almost verbatim, and our higraph-based Figure 14, which contains the same information.

A formalism that is very similar to that of E-R diagrams, and actually predated it by a number of years (see [40]), is that of *semantic*, or *associative, networks*. These graph-based structures are used widely in artificial intelligence for natural language processing and knowledge representation, and are discussed in numerous books and papers. (A good survey and history appears in [4], and more examples can be found in [6], [37], [44], and [50] and in the collection of papers in [17].) Semantic networks can actually be thought of as *concept-relationship diagrams*, with much of the research in the area concerned with the association of rich semantic meaning with the various types of nodes and edges. Here, too, is-a edges are used in abundance resulting in large, and at times incomprehensible, diagrams. Often, semantic networks contain more than one distinct type of is-a edges, corresponding to set inclusion, set membership, a physical "being-part-of" relationship, etc.[7] The way higraphs can be used here is exactly as in E-R diagrams; and the advantages become all the more significant if such different shades of structural is-a relationships can be made visually distinct (see the section called "Possible Variations on the Theme"). Clearly, it would be naive to claim that the profound problematics of knowledge representation can be overcome by diagrammatic considerations alone. Nevertheless, every little improvement helps.

In both E-R diagrams and semantic networks, people have observed that often the relationships, not only the entities and concepts, have to be stratified by levels of

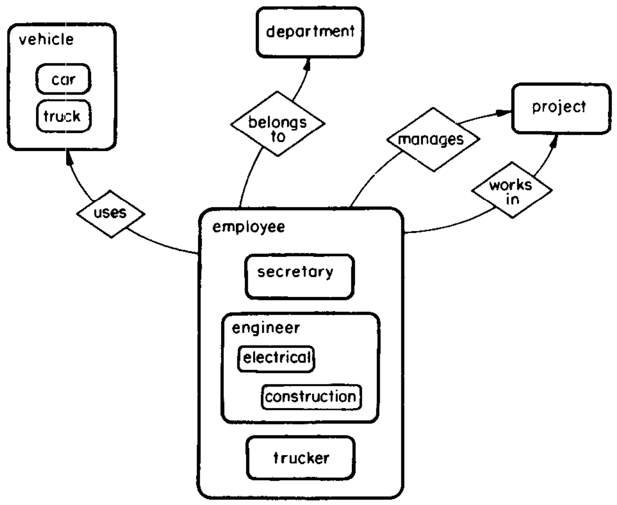


FIGURE 14. A Higraph-Based Version of Figure 13

[7] A variety of names have been attached to these, such as isa and inst in [6], SS and EL in [37] (standing for *is a, instance, subset*, and *element*, respectively), and many others elsewhere, such as a-kind-of, group-of, is-part-of, etc.

detail. This is typically done by considering the diamond-shaped relationship labels to be nodes of a second kind, and involving them also in structural is-a relationships with others. Although some people are opposed to this visual blurring of the distinction between entities and relationships, there is nothing to prevent those who are not from transferring this idea to the higraph framework. This would yield a blob structure also for the relationships, with the edges now serving to connect the entities and concepts to their relevant real, nonstructural relationships.

It is noteworthy that the area of the blobs in a higraph can be further exploited in these applications. Full E-R diagrams and semantic networks are typically laden with attributes, or properties, that are attached as additional "stump" nodes to the various entities. These attributes are often of the kind that are "inherited down" the is-a hierarchy, as the phrase goes. (In fact, there are many interesting issues associated with the very notion of inheritance; see [5], [45].) In a higraph-based representation, the area inside a blob would appear to be an ideal place to list, attach, or otherwise identify any properties, attributes, or explanations that are relevant to that blob and anything enclosed therein. Thus, simple inheritance is made possible quite naturally by the insideness approach to representing the subset relationship.

We should remark that some papers on semantic networks and the E-R model have indeed suggested the use of insideness and interblob edges to represent high-level entities and relationships, though the ideas do not seem to have been pursued to their full potential (see [10], [16], [25], [34], and [36]). Also, the idea of basing the decomposition of sets on Cartesian products and OR's is consistent with much of the literature on types. (For example, see [5] where these two features are captured by the notions of a *record* and a *variant*, respectively.)

Among the other graph-based formalisms for which higraphs appear to be useful are data-flow diagrams. A higraph-based version of such diagrams, called *activity-charts*, is one of the graphical languages supported by the STATEMATE system of i-Logix and is described in [24] and [28]. In activity charts the blobs denote functions, or activities, with the subset relation representing the subfunction relationship. The edges denote the possible flow of data. (Cartesian product is not used.) Consider the activity-chart of Figure 15, which is a simple part of the functional decomposition of an automatic teller machine. One of the edges therein means that the customer's account-number might possibly flow (following, perhaps, a read or write instruction) from the identify activity to the update-account activity, or to anywhere in the serve-customer activity, that is, to either (or all) of the deposit, withdraw, or balance-query subactivities. Another of the edges in Figure 15 means that the new amount with which the customer's balance should be adjusted might flow from any one of the deposit or withdraw activities to the update-account activity.

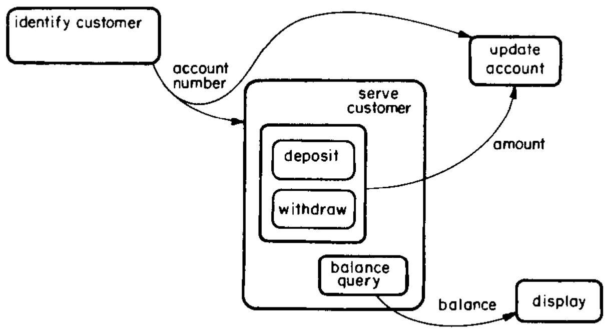


FIGURE 15. A Simple Activity Chart

Higraphs also form the basis of a recent paper [47], in which a visual language for specifying security constraints in operating systems is presented. The formalism represents access rights and exceptions thereof as distinct kinds of edges in a higraph, the blobs of which represent groups of users, files, and other entities. Cartesian product is used to represent the breakup of files into their components. Reference [47] also contains a number of interesting special-purpose extensions to the basic higraph formalism. Another use of higraph-like ideas appears in [32] and [38] in the form of proof diagrams for verifying concurrent programs, and there is a simple way of using higraphs as the basis of a hypertext system rather than conventional graph. In part, many issues that arise in the context of hypertext systems, such as multiple hierarchies, superconcepts, and composite nodes are treated naturally in the higraph formalism. (See [8].) One can also conceive of additional applications in visualizing interrupt-driven flowcharts and certain kinds of model-collapsing constructions in model theory.

STATECHARTS: A LESS OBVIOUS APPLICATION

The previous section notwithstanding, it would appear that the most beneficial application of higraphs lies in extending state-transition diagrams to obtain the *statecharts* of [21]. It was actually in the process of trying to formulate the underlying graphical concepts embodied in (the earlier) statecharts that higraphs emerged. This section contains a brief description of the statechart formalism; the reader is referred to [21] for further details.

To motivate the discussion, there appears to be agreement in the literature on software and systems engineering as to the existence of a major problem in the specification and design of large and complex *reactive systems*. A reactive system (see [22] and [39]), in contrast with a *transformational system*, is characterized by being event driven, continuously having to react to external and internal stimuli. Examples include telephones, communication networks, computer operating systems, avionics systems, VLSI circuits, and the man-machine interface of many kinds of ordinary software. The problem is rooted in the difficulty of describing reactive behavior in ways that are clear and realistic,

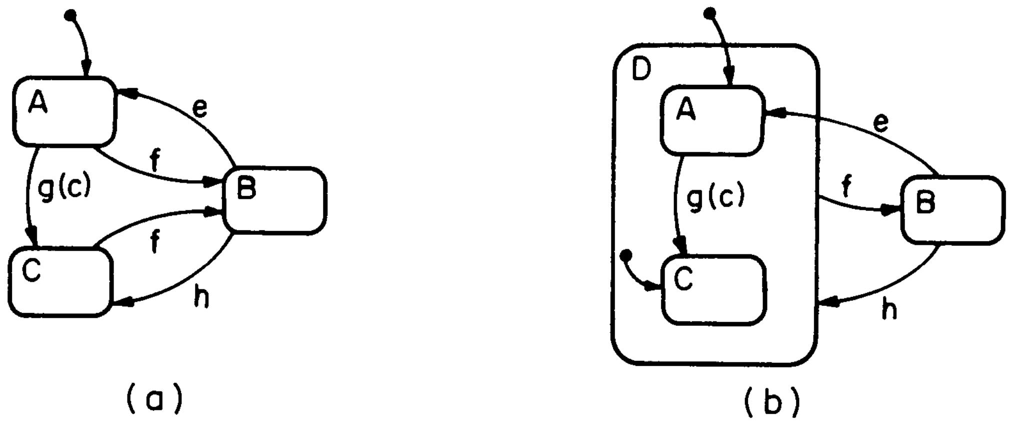


FIGURE 16. Depth in State Charts

and at the same time formal and rigorous, in order to be amenable to precise computerized analysis. The behavior of a reactive system is really the set of allowed sequences of input and output events, conditions, and actions, perhaps with some additional information such as timing constraints.

Most notable among the solutions proposed for this problem are Petri nets [41], communicating sequential processing (CSP) [26], the calculus of communicating systems (CCS) [35], the sequence diagrams of [51], ESTEREL [2], and temporal logic [39]. Statecharts constitute yet another attempt at solving this problem, but one that is aimed at reviving the classical formalism of finite-state machines (FSMs) and their visual counterpart, state-transition diagrams, trying to make them suitable for use in large and complex applications. Indeed, people working on the design of really complex systems have all but given up on the use of conventional FSMs and their state diagrams for several reasons:

(1) State diagrams are "flat." They provide no natural notion of depth, hierarchy, or modularity, and therefore do not support stepwise, top-down, or bottom-up development.

(2) State diagrams are uneconomical when it comes to transitions. An event that causes the very same transition from a large number of states, such as a high-level interrupt, must be attached to each of them separately resulting in an unnecessary multitude of arrows.

(3) State diagrams are extremely uneconomical, indeed quite infeasible, when it comes to states (at least when states are interpreted in the usual way as "snapshots" of the situation at a given point in time). As the system under description grows linearly, the number of states grows exponentially, and the conventional FSM formalism forces one to explicitly represent them all.

(4) Finally, state diagrams are inherently sequential in nature and do not cater for concurrency in a natural way.[8]

There have been attempts to remove some of these drawbacks, mostly by using various kinds of hierarchical or communicating state machines. Typically, however, these hierarchies provide little help in reducing the size of the resulting description, as they do not condense any information. Moreover, the communication between FSMs is usually one-to-one, being channel or processor based, and allows for only a single set of communicating machines on the highest level of the description. Furthermore, for the most part such extensions are not particularly diagrammatic in spirit, and hence one loses the advantages a visual medium might offer.

Statecharts are a higraph-based extension of standard state-transition diagrams, where the blobs represent states and arrows represent transitions. (For additional statechart features, the reader is again referred to [21].)[9] As to the basics, we might say that

state charts = state diagrams + depth
+ orthogonality + broadcast communication.

Depth is represented by the insideness of blobs, as illustrated in Figure 16, where 16b may replace 16a. The symbols e, f, g, and h stand for events that trigger the transitions, and the bracketed c is a condition. Thus, $g[c]$ triggers the transition from A to C if and when g occurs, but only if c is true at that time. The fact that A and C do not overlap and are completely inside D means that the latter is the *exclusive-or* (*XOR*) of the former, so that being in D is tantamount to being in either A or C, but not in both. The main point here is that the f-arrow, which leaves the contour of D, applies to both A and C, as in 16a. This simple higraph-based principle, when applied to large collections of states with many levels, helps overcome points (1) and (2) above (flatness and multilevel events). The idea of exploiting this kind of insideness in describing levels in a state-transition diagram appears also in [20]. It should be noted that the small *default arrows* depend on their

[8] Here, modeling a highly concurrent system by its global states only is considered unnatural.

[9] Some encouraging experimental evidence as to the appropriateness of statecharts for system description is discussed in [21, sect. 9].

encompassing blobs. In Figure 16a state *A* is singled out as being the default, or start state, of the three, a fact represented in 16b by the top default arrow. The bottom one, however, states that *C* is default among *A* and *C* if we are already in *D* and hence alleviates the need for continuing the *h*-arrow beyond *D*'s boundary.

Orthogonality is the dual of the *XOR* decomposition of states, in essence an *AND* decomposition, and is captured by the partitioning feature of higraphs, that is, by the unordered Cartesian product. In Figure 17b state *Y* consists of two *orthogonal components*, *A* and *D*, related by *AND*: To be in *Y* is tantamount to being in both *A* and *D*, and hence the two default arrows. The intended semantics of 17b is given by its equivalent "flat" version 17a, which represents a sort of automata product. Notice the simultaneity of transitions that takes place when event *e* occurs in state configuration (*B*, *F*), and the merging and splitting transitions that lead to and from *Y*. Note also the special condition [in(G)] attached to the *f*-transition from C, and the way it is reflected in Figure 17a. Figure 17 illustrates the heart of the exponential blowup problem, the number of states in the explicit version of *Y* being the product of the numbers of states in the orthogonal components of its higraph version. If orthogonality is used often and on many levels, the state explosion and sequentiality difficulties (points (3) and (4)) are also overcome in a reasonable way. This can be further observed by studying the examples and references in [21]).

Figures 16 and 17 do not contain any outputs, and hence, orthogonal components can synchronize so far only through common events (like *e* in Figure 17) and can affect each other only through [in(*state*)] conditions. A certain amount of subtlety is added to the way statecharts model concurrency by allowing *output* events. Here, statecharts can be viewed as an extension of Mealy machines (see [27]), since output events, which are called *actions*, can be attached optionally to the triggering event along a transition. In contrast with conventional Mealy machines, however, an action appearing along a transition in a statechart is not merely sent to the "outside world" as an output. Rather, it can affect the behavior of the state chart itself in orthogonal components. This is achieved by a simple broadcast mechanism: Just as the occurrence of an external event causes transitions in all components to which it is relevant (see Figure 17), if event *e* occurs and a transition labeled *e*/*f* is taken, the action *f* is immediately activated, and is regarded as a new event, possibly causing further transitions in other components.

Figure 18 shows a simple example of this. If we are in (*B*, *F*, *J*) and along comes the external event *m*, the next configuration will be (*C*, *G*, *I*), by virtue of *e* being generated in *H* and triggering the two transitions in components *A* and *D*. This is a *chain reaction* of length 2. If no external event *n* occurs, the new configuration will be (*B*, *E*, *J*), by virtue of a similar chain reaction of length 3.

This concludes our brief account of the basic features of statecharts, and we now illustrate the formalism with a rather simplified version of the digital watch described in [21]. The watch has four external control buttons, as well as a main display that can be used to

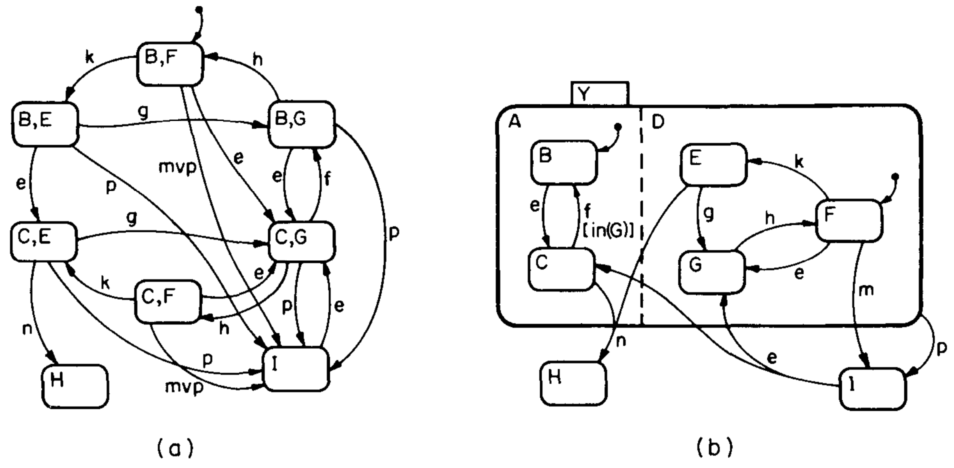


FIGURE 17. Orthogonality in State Charts

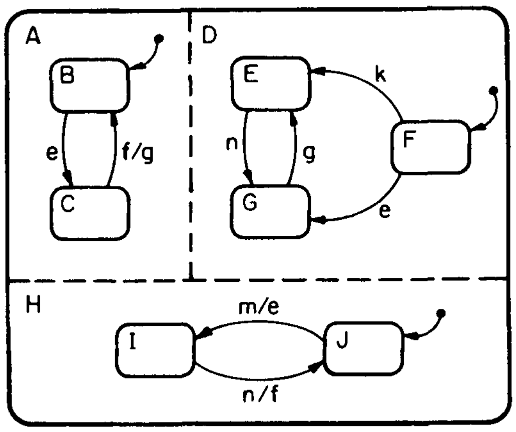


FIGURE 18. Broadcasting in State Charts

show the time (hour, minutes, and seconds) or the date (weekday, day of month, and month). It has a chime that can be enabled or disabled, beeping on the hour if enabled. It has an alarm that can also be enabled or disabled, and beeps for 2 minutes when the time in the alarm setting is reached unless any one of the buttons is pressed earlier. It has a stopwatch with two display modes (regular and lap), a light for illumination, and a weak-battery blinking indication.

Some of the external events relevant to the watch are a, b, c, and d, which signify the pressing of the four buttons, respectively, and b-up, for example, which signifies the release of button b. Another event we shall be using, 2-min, signifies that 2 minutes have elapsed since the last time a button was pressed. (We choose not to get involved here in a syntax for the event expressions themselves. In a language of compound events that includes a time-out construct, such as that of [24] and [28], this last event can be expressed easily.)

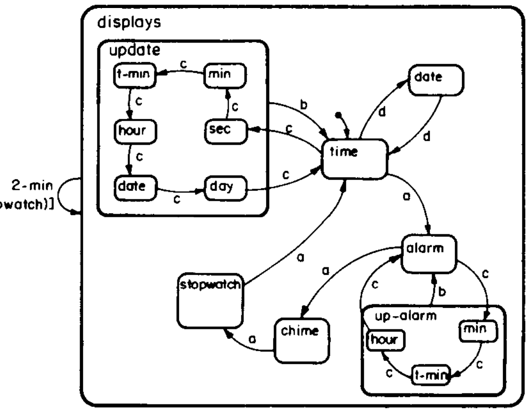


FIGURE 19. Part of the displays State in a Digital Watch

Statecharts can be used to describe the behavior of the watch in terms of its human interface; namely, how the user's operations, such as pressing buttons, influence things. It should be noted, however, that the descriptions that follow do not specify the activities carried out internally by the watch, only their control. Thus, nothing is said here about the time elapsing activity itself, or the technicalities of the beeping, the blinking, or the displays. These aspects of a system can be described using other means, and should be incorporated into the overall specification together with the statecharts. (See [24] for one approach to this incorporation.)

Figure 19 shows the basic displays state of the watch. Notice that time is the default state, and there is a cycle of pressings of a leading from time through the alarm, chime, and stopwatch states back to time. There is a general update state, and a special state for updating the alarm's internal setting. The 2-min event signifies return to time if 2 minutes have elapsed in any state other than stopwatch and no button has been pressed.

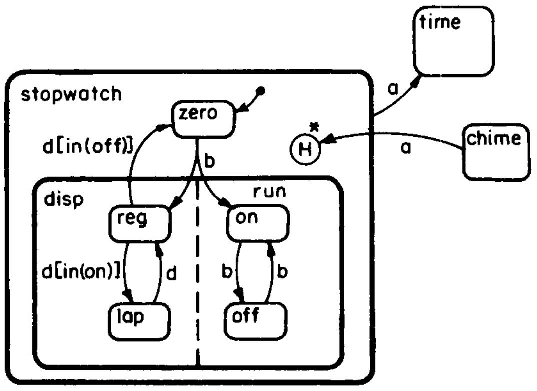


FIGURE 20. The stopwatch State

The specification of the watch contains examples of orthogonal states on various levels. We should first consider the stopwatch state, detailed in Figure 20. It has two substates, zero and {disp, run}, the first being the default. Pressing b takes the stopwatch from the former to the latter causing it to start running with a regular display. Repeatedly pressing b causes it to stop and start alternately. Pressing d can be seen to cause the display to switch to lap and back to reg, or to leave the orthogonal state and return to zero depending, as illustrated, on the present state configuration. The encircled and starred H is one of the additional notations described in [21], and prescribes that, upon entering stopwatch from chime by pressing a, the state actually entered will be the one in which the system was in most recently. Thus, we are entering the stopwatch state by "history"—hence, the H. The default will be used if this is the first time stopwatch is entered, or if the history has been cleared.

The description of the high levels of the watch also uses orthogonality. In Figure 21 the watch is specified as being either dead or alive, with the latter consisting of five orthogonal components. (Notice where the

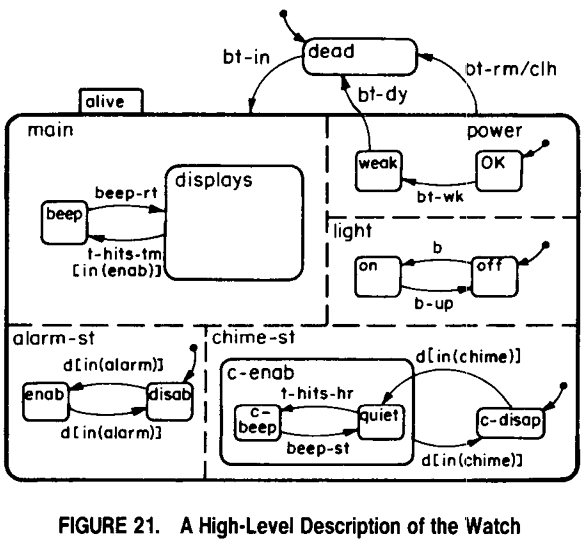


FIGURE 21. A High-Level Description of the Watch

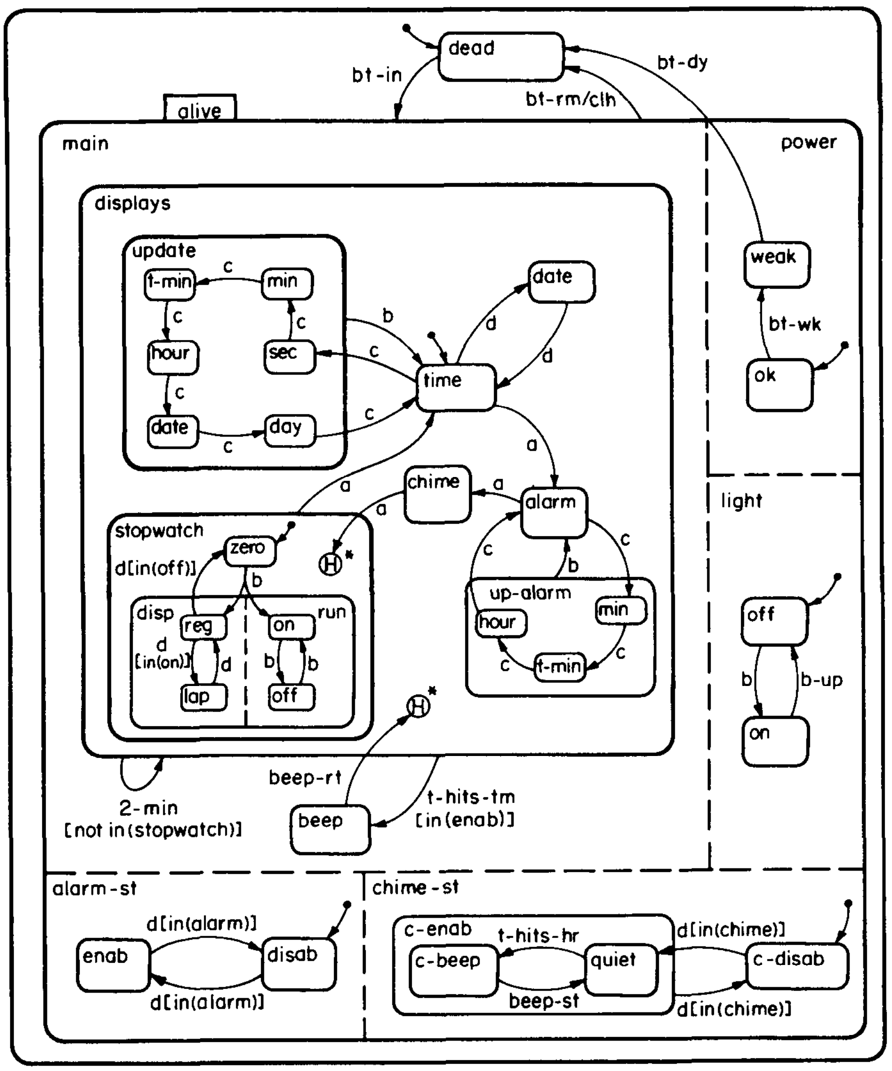


FIGURE 22. A State Chart for the Digital Watch

displays state fits in.) In this figure the events bt-in, bt-rm, bt-dy and bt-wk signify, respectively, the insertion, removal, expiration, and weakening (below a certain level) of the battery. We use t-hits-tm to signify that the internal time of the watch has reached the internal time setting of the alarm, and t-hits-hr to signify that it has reached a whole hour. Also, beep-rt occurs when either any button is pressed or 2 minutes have elapsed since entering beep, and beep-st occurs 2 seconds after entering c-beep. (As mentioned, these events should also be written formally as compound event expressions in a language involving time-outs, disjunctions, and so on; see [28].)

The first of the five components in Figure 21, main, specifies the transitions between displaying and beeping, where displays is simply the state described earlier (see Figure 19). (In actuality, the displaying activities themselves do not shut off when the watch is beeping, but cannot be changed until control returns to the displays state.) The alarm-st component describes the status of the alarm, specifying that it can be changed using d when control is in the alarm display state. The chime-st state is similar, with the additional provision for beeping on the hour given within. The power state is self-explanatory, where the activity that would take place in the weak state would involve the displays blinking frantically.

In considering the innocent-looking light state, the default is off, and depressing and releasing b cause the light to switch alternately between on and off. What is interesting is the effect these actions might have elsewhere. If the entire statechart for the parts of the watch described so far is contemplated (see Figure 22), one realizes that pressing b for illumination has significant side effects: It will cause a return from an update state if we happen to be in one, the stopping of the alarm if it happens to be beeping, and a change in the stopwatch's behavior if we happen to be working with it. Conversely, if we use b in displays for any one of these things the light will go on, whether we like it or not. These seeming anomalies are all a result of the fact that the light component is orthogonal to the main component, meaning that its scope is very broad. One can imagine a far more humble light component, applicable only in the time and date states, which would not cause any of these problems. Its specification could be carried out by attaching it orthogonally, not to main, but to a new state surrounding time and date, as in Figure 23.

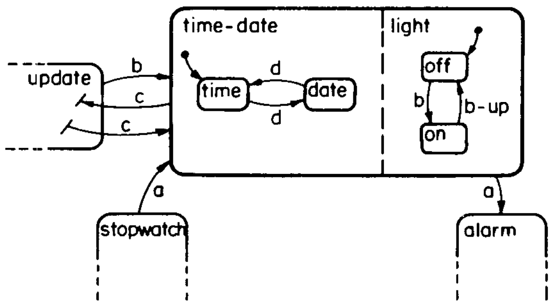


FIGURE 23. A Smaller Scope for the Light

As mentioned earlier, this section has only described the "no-frills" version of the statecharts. A more complete treatment appears in [21], and a formal syntax and semantics appear in [23]. The reader may have noticed that we have not used intersecting states in the statecharts. While intersecting blobs in higraphs do not cause any serious semantic problems (see the appendix), intersecting states in state charts do. In fact, since not all syntactically legal higraphs make sense as statecharts, it is not even clear how to define an appropriate syntax for statecharts with intersecting states (see [21, sect. 6.2]). A preliminary approach to these problems appears in [29].

POSSIBLE VARIATIONS ON THE THEME

The higraph formalism can be enriched and extended in various ways. We shall point to a few of these possibilities briefly and informally.

At times it becomes useful to base a formalism on a three-valued, rather than a two-valued, underlying model. For example, in certain uses of graphs in databases and artificial intelligence there arises a need to state not only that a certain relationship R holds or does not hold between two objects, but also to capture the situation whereby we do not know which of these is the case. One possibility is to reinterpret the absence of an R arrow as denoting the don't-known situation, and have a new kind of arrow representing the *negative information* that R definitely does *not* hold. This simple idea can be adopted in higraphs too, as in Figure 24, which is suppose to indicate that R holds between A and B and does not hold between B and C, and that all

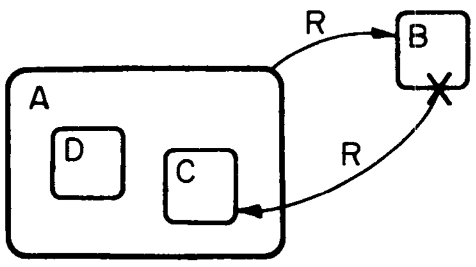


FIGURE 24. Negative Arrows

other possibilities (including whether or not R holds between C and B)[10] are left open.

Often a don't-know option is needed not only for arrows, but for blobs as well. That is, we might want to represent uncertainty as to the presence or absence of identifiable sets, rather than relationships. Accordingly, we can use a new blob notation (e.g., one with a dashed contour) to denote a set that we are not sure actually exists (here one assumes that all regular blobs stand for nonempty sets). Figure 25 asserts our uncertainty as to whether $A - B$ is empty or not, and also states that if it is not empty then the difference is called E and is related to F via relationship R.

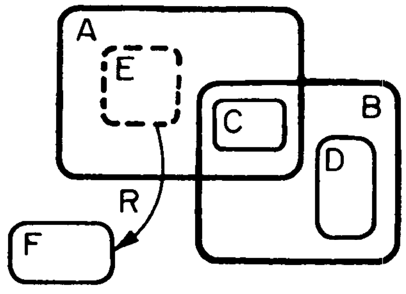


FIGURE 25. "Not-Quite-Sure" Blobs

When higraphs are used in practice (see [21], [24], [28], and [47]), it is useful to be able to "zoom out" of a particular view, suppressing low-level details. A good example would be going from Figure 22, the detailed state-chart description of the watch, to the less detailed Figure 21. In such cases there arises a problem with edges connected to subblobs that are omitted from the new, less detailed view. If we decide to zoom out of the likes of Figure 26 by suppressing blobs B and C, it might be a mistake to consider Figure 27a as the correct new version, since the two are clearly inconsistent. Figure 27b is better, with its stubs that represent relationships to unspecified subblobs. For example, since a statechart arrow whose target is a high-level state A prescribes entrance to none other than the default substate

[10] This is not determined by the arrow from A to B, since, as discussed earlier, the fact that R holds between A and B says nothing about what the case is for A's subsets.

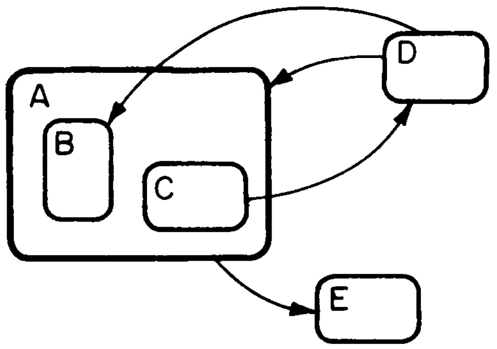


FIGURE 26. Another Simple Higraph

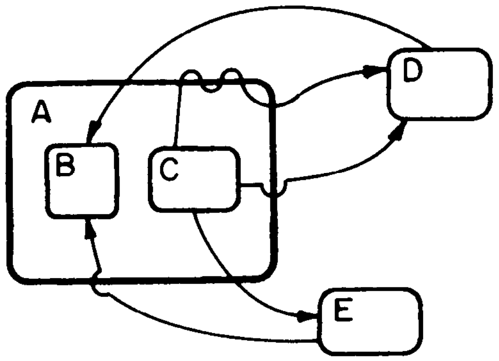


FIGURE 27. Two Possible Zoom Outs for Figure 26

of *A*, Figure 21 is somewhat inconsistent with Figure 22. In the present context, a better version would have shown the `beep-rt` arrow crossing the contour of the `displays` state and ending with a stub indicating entrance to a substate (as of now unspecified) that is possibly different from the default substate, `time`.

One weakness of the higraph formalism is its inability to specify both set inclusion and set membership. We have chosen to adopt the former as the meaning of blob enclosure, although we could probably have chosen the latter too without causing too many problems. This weakness is all the more apparent when higraphs are contrasted with their graph-based equivalents, in which set inclusion is depicted by `is-a` edges (see Figure 10). In the latter, one need only use an additional type of edge, labeled `elmnt-of`, for instance, to be able to represent set membership. We would like to claim that this is not much more than a notational problem that requires a topo-visual way of distinguishing between two different kinds of insideness. Most of the solutions to this notational problem that come to mind are somewhat unsatisfactory, with the exception of the one that calls for a three-dimensional basis for higraphs, in which the third dimension is responsible for such distinctions (e.g., by having set inclusion take place in the same plane and set membership be reflected by different levels of planes).[11]

An additional possible extension to higraphs is to make arrows mean more than a simple connection between source and target. (We are assuming ordinary directed binary edges here, not, say, hyperedges.) Since higraph arrows in general cut across blob contours, we might want to say something more about the *sequence* of crossovers that the edge takes on its way from the source to the target. This can be achieved trivially by drawing the arrow through the appropriate contours in the desired order (assuming this order is indeed possible, given the basic topology of the blobs). The interesting case occurs when we want to omit from such a sequence one or more of the contours that, topologically speaking, must be crossed by any line from the source to the target. We would like the *D*-to-*B* arrow in Figure 26, for example, to enter *B*, but *not* to enter *A* in the process. State charts with intersections give rise to one interesting motivation for such cases, whereby one wants the system to enter only one of two intersecting

[11] Visual formalisms that are predominantly two-dimensional in nature, but make some use of a third dimension, are far from being out of the question, even if we are not willing to wait for quality holographic workstations to show up. If all we need, as in this case, is the ability to tell when two nested blobs are on the same plane or not, then a simple graphical simulation of a dynamic left-right shift in point of view would do the job.

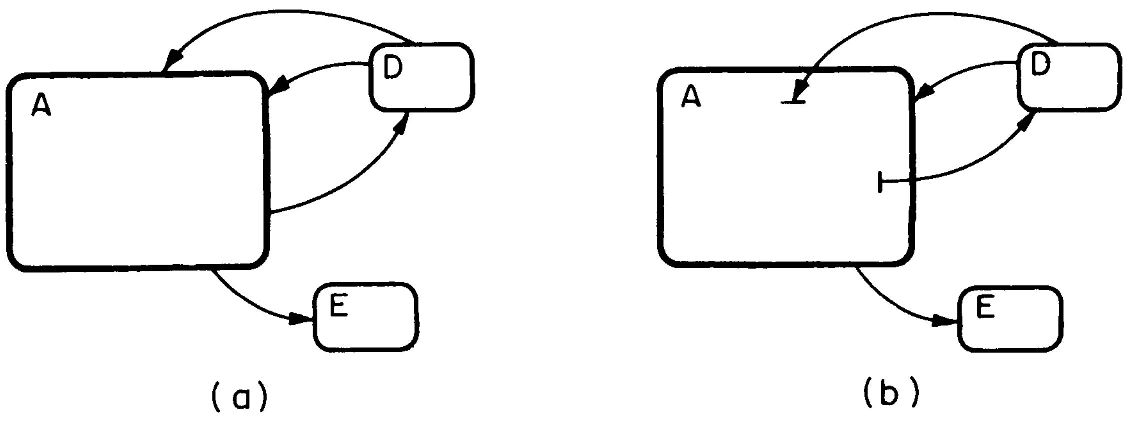


FIGURE 28. Skipping and Multiple Crossovers

states; again, the reader is referred to [21, sec. 6.2] for details. This richer notion of an edge can be represented visually by simply allowing arrows to skip edges as in Figure 28. Multiple crossovers, if desired, can also be represented as illustrated in the figure. Clearly, the formal semantics would be more elaborate, since a finite sequence of blobs, rather than an ordered pair, is the interpretation of a directed edge, and a finite set thereof, rather than an unordered pair, is the interpretation of an undirected edge.

CONCLUSION AND FUTURE WORK

Higraphs seem to give rise to several interesting mathematical notions adapted to a large extent from graphs and hypergraphs. For example, one can provide reasonable definitions of connectivity, transitive closure, planarity, and acyclicity in higraphs, as well as a couple of different notions of "hitrees." For each of these, we may ask for upper and lower bounds on the computational complexity of the corresponding algorithmic problems. In some cases algorithms and bounds can be carried over from the work on graphs and hypergraphs, but one gets the feeling that in other cases these bounds can be improved by utilizing the special structure of higraphs. Some of these algorithmic problems have indeed arisen during the implementation of the STATEMATE system [24, 28], which supports three higraph-based formalisms. It would appear that the algorithmics of higraphs forms a fruitful avenue for further research.

The main thesis underlying this paper is that the intricate nature of a variety of computer-related systems and situations can, and in our opinion should, be represented by *visual formalisms*: visual, because they are to be generated, comprehended, and communicated by humans; and formal, because they are to be manipulated, maintained, and analyzed by computers. (This thesis is consistent with the study in [9], which argues for a more visual, nonverbal approach toward mathematics.)

Part of our motivation in stressing this point, despite the fact that it might appear to be so obvious, is the rather different approach that one occasionally finds elsewhere. For example, [33] is a compendium of many computer-related diagrammatic methods (virtually all of which are based on graphs). In our opinion, [33] is quite inadequate, since it accepts the *visual*, but apparently rejects the *formal*. For the most part, the methods and languages appearing in [33] are described in a manner that is devoid of semantics, and can therefore be used at best as informal aids when working with some other, hopefully more rigorous, nonvisual medium.

One of the implicit points we have tried to make in this article is that a considerable amount of mileage can be gotten out of basing such formalisms on a small number of simple diagrammatic notions, first and foremost among which are those that are topological in nature, not geometric. A lot can be gained by using topo-visual formalisms based on insideness, connectedness, and partitioning, with the semantics as given here, before one attempts to attach special significance to, for example, shapes, colors, and sizes.

We are entirely convinced the future is "visual." We believe that in the next few years many more of our daily technical and scientific chores will be carried out visually, and graphical facilities will be far better and cheaper than today's. The languages and approaches we shall be using in doing so will not be merely iconic in nature (e.g., using the picture of a trash can to denote garbage collection), but inherently diagrammatic in a conceptual way, perhaps also three-dimensional and/or animated. They will be designed to encourage visual modes of thinking when tackling systems of ever-increasing complexity, and will exploit and extend the use of our own wonderful visual system in many of our intellectual activities.

APPENDIX. Formal Definition of Higraphs

In what follows we present a formal (nongraphical) syntax and semantics for higraphs with simple binary directed edges. The reader should have no difficulty in extending the edge set E to represent, say, hyperedges.

A *higraph* is a quadruple

$$H = (B, \sigma, \pi, E),$$

where B is a finite set of elements, called *blobs*, and E, the set of *edges*, is a binary relation on B:

$$E \subseteq B \times B.$$

The *subblob* function σ is defined as

$$\sigma: B \longrightarrow 2^B.$$

It assigns to each blob $x \in B$ its set $\sigma(x)$ of subblobs and is restricted to being cycle free. Thus, if we define

$$\sigma^0(x) = \{x\}, \qquad \sigma^{i+1}(x) = \bigcup_{y \in \sigma^i(x)} \sigma(y),$$

$$\text{and} \qquad \sigma^+(x) = \bigcup_{i=1}^{\infty} \sigma^i(x),$$

then σ is restricted so that $x \notin \sigma^+(x)$.

The *partitioning* function π is defined as

$$\pi: B \longrightarrow 2^{B \times B},$$

associating with each blob $x \in B$ some equivalence relation $\pi(x)$ on the set of subblobs, $\sigma(x)$. This is really just a rigorous way of specifying the breakup of x into its orthogonal components, which are now defined simply to be the equivalence classes induced by the relation $\pi(x)$. Indeed, for $x \in B$ let us denote these classes by $\pi_1(x), \ldots, \pi_{k_x}(x)$. For the orthogonal division into components to be representable graphically (and in order to make the semantics cleaner), we shall require that blobs in different orthogonal components of x are disjoint. Formally, for each x we require that no two elements y and z of $\sigma(x)$ can intersect—that is, can satisfy $\sigma^+(y) \cap \sigma^+(z) \neq \emptyset$—unless they are in the same orthogonal component—that is, unless the relation $\pi(x)$ renders them equivalent. Clearly, $k_x = 1$ means x is not partitioned into components at all.

This concludes the syntax of higraphs; now for the semantics. Two notations are useful. Given a higraph H, define the set of *atomic blobs* to be

$$A = \{x \in B \mid \sigma(x) = \emptyset\}.$$

(Obviously, the finiteness of B and the cycle-freeness restriction on σ imply A is nonempty.) The *unordered Cartesian product* of two sets S and T is defined as

$$S \otimes T = \{\{s, t\} \mid s \in S,\ t \in T\}.$$

Given a higraph H, a *model* for H is a pair

$$M = (D, \mu),$$

where D is a set of unstructured elements[12] called the *domain* of the model M, and μ assigns disjoint subsets of D to the atomic blobs of H. Thus,

$$\mu\colon A \to 2^D,$$

where if $x \neq y$ then $\mu(x) \cap \mu(y) = \emptyset$. We now have to show how to extend the association of atomic blobs with sets over D to an association of all blobs with more complex objects over D. Accordingly, extend μ by defining, inductively, for each $x \in B$,

$$\mu(x) = \bigotimes_{i=1}^{k_x} \left(\bigcup_{y \in \pi_i(x)} \mu(y) \right),$$

the intuition being that to calculate the semantics of a blob x we form the unordered Cartesian product of the meanings of its orthogonal components, each of which, in turn, is simply the union of the meanings of its constituent blobs. In particular, of course, if $k_x = 1$, no product is taken, and we really have

$$\mu(x) = \bigcup_{y \in \sigma(x)} \mu(y),$$

as expected.

To complete the semantics, note that the edge set E induces a semantic relation E_M on the $\mu(x)$s, defined by

$$(\mu(x), \mu(y)) \in E_M \qquad \text{iff} \quad (x, y) \in E.$$

[12] We want to avoid situations in which, say, x and $\{x\}$ are both elements of D.

Acknowledgments. Thanks are due to Ton Kalker, Doug Tygar, and Jeanette Wing for comments on the appendix, and to an anonymous referee for a very detailed and thoughtful report.

REFERENCES

1. Berge, C. *Graphs and Hypergraphs.* North-Holland, Amsterdam, 1973.
2. Berry, G., and Cosserat, I. The ESTEREL synchronous programming language and its mathematical semantics. In *Seminar on Concurrency,* S. Brookes and G. Winskel, Eds. Lecture Notes in Computer Science, vol. 197. Springer-Verlag, New York, 1985, pp. 389–448.
3. Biggs, N.L., Lloyd, E.K., and Wilson, R.J. *Graph Theory: 1736-1936.* Clarendon Press, Oxford, 1976.
4. Brachman, R.J. On the epistemological status of semantic networks. In *Associative Networks: Representation and Use of Knowledge by Computer,* N.V. Findler, Ed. Academic Press, New York, 1979, pp. 3–50.
5. Cardelli, L.A. Semantics of multiple inheritance in semantics of data types. Kahn, G. et al. Lecture Notes in Computer Science. vol. 173, Springer-Verlag, 1984, pp. 51–67.
6. Charniak, E., and McDermott, D. *Introduction to Artificial Intelligence.* Addison-Wesley, Reading, Mass., 1985.
7. Chen, P.P.-S. The entity-relationship model—toward a unified view of data. *ACM Trans. Database Syst. 1,* 1 (Mar. 1976), 9–36.
8. Conklin, J. Hypertext: An introduction and survey. *IEEE Computer 20,* 9 (Sept. 1987), 17–41.
9. Davis, P.J., Anderson, J.A. Nonanalytic aspects on mathematics and their implication on research and education. *SIAM Review 21,* 1 (Jan. 1979), 112–127.
10. dos Santos, C.S., Neuhold, E.J., and Furtado, A.L. A data type approach to the entity-relationship model. In *Entity-Relationship Approach to Systems Analysis and Design,* P.P. Chen, Ed. North-Holland, Amsterdam, 1980, pp. 103–119.
11. Dugundji, J. *Topology.* Allyn and Bacon, Boston, Mass., 1966.
12. Euler, L. Solutio problematis ad geometriam situs pertinentis. *Comm. Acad. Sci. Imp. Petropol. 8* (1736), 128–140.
13. Euler, L. *Lettres à une Princesse d'Allemagne.* Vol. 2. 1772 (letters 102–108).
14. Fagin, R. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM 30,* 3 (July 1983), 514–550.
15. Fagin, R., Mendelzon, A., and Ullman, J. A simplified universal relation assumption and its properties. *ACM Trans. Database Syst. 7,* 3 (Sept. 1982), 343–360.
16. al-Fedaghi, S.S. An entity-relationship approach to modelling petroleum engineering database. In *Entity-Relationship Approach to Software Engineering,* C.G. Davis et al., Eds. Elsevier Science Publishers, Amsterdam, 1983, pp. 761–779.
17. Findler, N.V., Ed. *Associative Networks: Representation and Use of Knowledge by Computer.* Academic Press, New York, 1979.
18. Fitter, M., and Green, T.R.G. When do diagrams make good computer languages? *Int. J. Man-Mach. Stud. 11,* 2 (March 1979), 235–261.
19. Gardner, M. *Logic Machines and Diagrams.* 2nd ed. University of Chicago Press, Chicago, Ill., 1982.
20. Green, T.R. Pictures of programs and other processes, or how to do things with lines. *Behav. Inf. Technol. 1,* 1 (1982), 3–36.
21. Harel, D. Statecharts: A visual formalism for complex systems. *Sci. Comput. Program. 8,* 3 (June 1987), 231–274.
22. Harel, D., and Pnueli, A. On the development of reactive systems. In *Logics and Models of Concurrent Systems,* NATO, ASI Series, vol. 13, K.R. Apt, Ed. Springer-Verlag, New York, 1985, pp. 477–498.
23. Harel, D., Pnueli, A., Schmidt, J.P., and Sherman, R. On the formal semantics of statecharts. In *Proceedings of the 2nd IEEE Symposium on Logic in Computer Science* (Ithaca, N.Y., June 22–24). IEEE Press, New York, 1987, pp. 54–64.
24. Harel, D., Lachover, H., Naamad, A., Pnueli, A., Politi, M., Sherman, R., and Shtul-Trauring, A. STATEMENT: A working environment for the development of complex reactive systems. In *Proceedings of the Tenth IEEE International Conference on Software Engineering* (Singapore, April 13–15). IEEE Press, New York, 1988.
25. Hendrix, G.G. Expanding the utility of semantic networks through partitioning. In *Proceedings of the 4th International Conference on Artificial Intelligence* (Tbilisi, Georgia, USSR, Sept. 3–8). International Joint Council on Artificial Intelligence, Cambridge, Mass., 1975, pp. 115–121.
26. Hoare, C.A.R. Communicating sequential processes. *Commun. ACM 21,* 8 (Aug. 1978), 666–677.
27. Hopcroft, J.E., and Ullman, J.D. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, Mass., 1979.
28. i-Logic. The languages of STATEMATE. Tech. Rep., i-Logix, Burlington, Mass., 1987.

29. Kahana, C.A. Statecharts with overlapping states. M.S. thesis, Dept. of Mathematics and Computer Science, Bar-Ilan University, Ramat Gan, Israel, 1986 (in Hebrew).
30. Lefschetz, S. *Introduction to Topology.* Princeton University Press, Princeton, N.J., 1949.
31. Maier, D., and Ullman, J.D. Connections in acyclic hypergraphs. In *Proceedings of the ACM Symposium on Database Systems* (Los Angeles, Calif., March 29–31). ACM, New York, 1982, pp. 34–39.
32. Manna, Z., and Pnueli, A. Specification and verification of concurrent programs by ∀-automata. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages* (Munich). ACM, New York, 1987, pp. 1–12.
33. Martin, J., and McClure, C. *Diagramming Techniques for Analysts and Programmers.* Prentice-Hall, Englewood Cliffs, N.J., 1985.
34. McSkimin, J.R., and Minker, J. A predicate calculus based semantic network for deductive searching. In *Associative Networks: Representation and Use of Knowledge by Computer*, N.V. Findler, Ed. Academic Press, New York, 1979, pp. 205–238.
35. Milner, R. *A Calculus of Communicating Systems.* Lecture Notes in Computer Science, vol. 92. Springer-Verlag, New York, 1980.
36. Nakano, R. Integrity checking in a logic-oriented ER model. In *Entity-Relationship Approach to Software Engineering*, C.G. Davis et al., Eds. Elsevier Science Publishers, Amsterdam, 1983, pp. 551–564.
37. Nilsson, N.J. *Principles of Artificial Intelligence.* Tioga, Palo Alto, Calif., 1980.
38. Owicki, S., and Lamport, L. Proving liveness properties of concurrent programs. *ACM Trans. Program. Lang. Syst. 4*, 3 (July 1982), 455–495.
39. Pnueli, A. Applications of temporal logic to the specification and verification of reactive systems: A survey of current trends. In *Current Trends in Concurrency*, J. W. de Bakker et al., Eds. Lecture Notes in Computer Science, vol. 224, Springer-Verlag, New York, 1986, pp. 510–584.
40. Quillian, M.R. Semantic memory. In *Semantic Information Processing*, M. Minsky, Ed. MIT Press, Cambridge, Mass., 1968, pp. 227–270.
41. Reisig, W. *Petri Nets: An Introduction.* Springer-Verlag, Berlin, 1985.
42. Schiffner, G., and Schuermann, P. Multiple views and abstractions with an extended-entity-relationship model. *Comput. Lang.* 4, 3/4 (1979), 139–154.
43. Schmid, C.F. *Statistical Graphics: Design Principles and Practices.* Wiley, New York, 1983.
44. Shapiro, S.C. A net structure for semantic information storage, deduction, and retrieval. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence.* 1971, pp. 512–523.
45. Touretzky, D.S. *The Mathematics of Inheritance Systems.* Pitman, London, and Morgan Kaufmann, Los Altos, Calif. 1986.
46. Tufte, E.R. *The Visual Display of Quantitative Information.* Graphics Press, Cheshire, Conn., 1983.
47. Tygar, J.D., and Wing, J.M. Visual specification of security constraints. In *The IEEE Workshop on Visual Languages* (Linköping, Sweden, Aug. 19–21). IEEE Press, New York, 1987.
48. Venn, J. On the diagrammatic and mechanical representation of propositions and reasonings. *Phil. Mag.* (1880), 123.
49. Venn, J. *Symbolic Logic.* 2nd ed. London, 1894. (Reprinted by Chelsea, Bronx, N.Y., 1971.)
50. Woods, W.A. What's in a link? Foundations for semantic networks. In *Representation and Understanding*, D.G. Bobrow and A.M. Collins, Eds. Academic Press, New York, 1975, pp. 35–82.
51. Zave, P. A distributed alternative, to finite-state-machine specifications. *ACM Trans. Program. Lang. Syst. 7*, 1 (Jan. 1985), 10–36.

CR Categories and Subject Descriptors: C.0 [**Computer Systems Organization**]: General—*systems specification methodology*; C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems—*real-time systems*; D.0 [**Software**]: General; D.2.1 [**Software Engineering**]: Requirements/Specifications; D.2.2]: Tools and Techniques; D.2.10 [**Software Engineering**]: Design; E.0 [**Data**]: General; E.1 [**Data**]: Data Structures—*graphs*; F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*automata*; H.1.0 [**Models and Principles**]: General; I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods—*semantic networks*

General Terms: Design, Languages, Theory

Additional Key Words and Phrases: Higraph, reactive systems, state charts, visual formalisms

Author's Present Address: David Harel, Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, Israel 76100.