On the Algorithmics of Higraphs

O. Grossman & D. Harel

Technical Report CS97–15

September 1997

THE WEIZMANN INSTITUTE OF SCIENCE

DEPARTMENT OF APPLIED MATHEMATICS AND COMPUTER SCIENCE

76100 REHOVOT ISRAEL

On the Algorithmics of Higraphs*

Ornit Grossman and David Harel†
Dept. of Applied Mathematics and Computer Science
The Weizmann Institute of Science, Rehovot, Israel

Abstract

This paper initiates an algorithmic investigation of *higraphs*, a formalism proposed in 1988 that combines and extends graphs and Euler/Venn diagrams. We first provide higraphs with a syntax and semantics that minimize ambiguities in the interpretation. We then consider some basic properties of higraphs, such as shortest paths, Hamiltonian cycles, bipartition, and minimum cover, and seek algorithms to test for them. In some cases, like shortest paths, the algorithms and the bounds from graphs can be generalized to higraphs, but in others, such as Hamiltonian cycles, the situation is not as clear.

1 Introduction

Higraphs, a combination and extension of graphs and Euler/Venn diagrams, were defined in [4]. They are formed by modifying Euler/Venn diagrams somewhat, extending them to represent Cartesian products, and connecting the resulting 'blobs' by edges or hyperedges. Higraphs enable compact representations of sets of elements related set-theoretically, together with some special relation on them provided by the edges. Several applications of higraphs are discussed in [4] (see, e.g., [2, 8, 10, 11]), the main one (which was also the motivation for defining higraphs) being the language of statecharts [3]. The Cartesian products — which in statecharts depict orthogonal, or concurrent states — prevent certain representations from growing exponentially in size.

Despite these applications, very little work has been carried out on the algorithmic properties of higraphs. One effort in this direction involves algorithms for detecting ambiguity in instance pictures, which are higraph-based specifications of file system security constraints [10, 11]. Also, some algorithmic problems have arisen during the implementation of the STATEMATE and Rhapsody systems [8, 5], which support a number of higraph-based formalisms, including statecharts.

In the present paper we would like to initiate a wider study of algorithms on higraphs. In order to do so, higraphs must first be provided with a syntax and semantics that minimize the possibilities for ambiguities in their interpretation. In Section 2 we discuss a number of possibilities for these, and choose the one that seems most consistent and intuitive. This definition views higraphs as an extension of ordinary graphs, but one that can be exponentially more succinct.

*Parts of this work were supported by grants from the Israeli Ministry of Science and Technology.

†This author holds the William Sussman chair in mathematics. Parts of this author's work were carried out during a summer spent at the Center for Excellence in Space Data and Information Sciences (CESDIS) at NASA Goddard Space Flight Center, Greenbelt, MD.

Sections 3, 4, 5 and 6 define some basic properties of higraphs, including shortest paths between configurations, Hamiltonian cycles, and bipartition. We seek efficient algorithms for detecting these properties. As anticipated in [4], in some cases the algorithms and bounds carry over with few changes from graphs, but in other cases the two formalisms behave differently, due to the richer structure of higraphs.

Section 7 deals with compact representation via a minimum covering problem, which is relevant to higraphs but not to graphs.

2 Definitions

In [4] a simple set-theoretic interpretation of higraphs was given, which was sufficiently open-ended as to enable a variety of different applications. When blobs, the node-like elements of a higraph, are interpreted as states in the behavioral description of a system and edges are interpreted as transitions, one is led to the language of statecharts [3]. Other interpretations of blobs and edges involve entities and relationships in database systems [3], object classes and role relationships in object-oriented analysis [5], or file clusters and access rights in system security [10, 11].

Since our goal is to produce results that are useful for most applications of higraphs, we would like the syntax and set-theoretic semantics of higraphs to preserve the characteristics of general sets and graphs as much as possible. A useful approach views higraphs as an extension of ordinary graphs by AND/OR decomposition of vertices. The two essential ideas which enable this extension are the provision for depth, or hierarchy (depicted by encapsulation), and the notion of orthogonality, or Cartesian product (depicted by juxtaposed partitioning using dashed lines). In the spirit of [3], we may write:

$$\text{higraphs} = \text{graphs} + \text{depth} + \text{orthogonality}$$

These additions enrich the succinctness of higraphs exponentially relative to graphs. In fact, each of the two alone provides super-polynomial savings in the size of the description: while n vertices in a graph can represent n different entities, a higraph can represent $\Theta(2^n)$ entities with n blobs using orthogonality, and we shall see that it can also represent $\Theta(\sqrt{n}^{\sqrt{n}})$ entities with n blobs using depth and blob intersection. Our 'entities' will be appropriately defined sets of blobs, which, after [3, 9, 7], are called *configurations*.

Having defined the configurations of a higraph H, we can talk about its semantics in terms of the *induced graph* G_H. The vertices of G_H are taken to be the configurations of H, and there is an edge between two vertices in G_H if and only if there is an 'appropriate' edge in H. There are several possibilities for defining this, and we justify the one we adopt on the grounds that other conceivable possibilities have unacceptable disadvantages. This approach to semantics enables us to represent higraphs by means of ordinary graphs, and thus to compare the algorithmic properties of a higraph with those of its induced graph. The exponential gap between the two representations raises the interesting issue of whether the properties we consider turn out to require exponentially worse algorithms for higraphs than for graphs.

Throughout this paper we assume that higraphs have simple binary edges only (i.e., not hyperedges as in [4]), and we assume that between two blobs there can be at most one edge in each direction. We concentrate on directed higraphs, though most of our results hold for the undirected case too. We also restrict ourselves to higraphs in which an intersection of two different blobs is contained in at most one orthogonal component of each of the blobs.

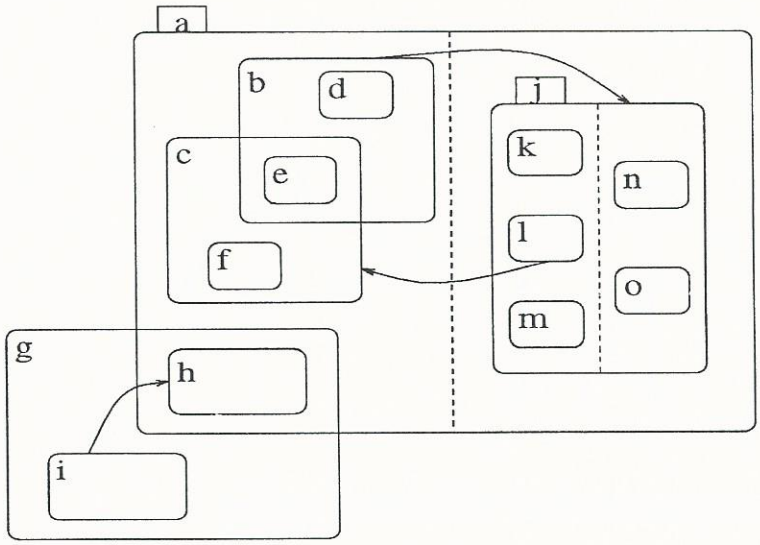

Figure 1: A simple higraph

2.1 Syntax

Definition 1 *A* higraph H *is a quadruple* (B, E, ρ, π), *where* B *is a set, the elements of which are called* blobs, *and* E, *the set of* edges, *is a subset of* $B \times B$. *In addition,* $\rho : B \to 2^B$ *is the* hierarchy function, *defining the direct descendants of a blob, and* $\pi : B \to 2^{B \times B}$ *is the* partitioning function, *explained below.*

We let ρ^* and ρ^+ be the reflexive transitive closure, and the irreflexive transitive closure, respectively of ρ. We require that ρ^+ be acyclic, so that the *containment graph* (B, ρ'), where $\rho' = \{(b, c) : c \in \rho(b)\}$, is a DAG.

The partitioning function π is constrained to associate with each blob $b \in B$ an equivalence relation $\pi(b)$ on the set $\rho(b)$ of b's descendants. This specifies the breakup of b into its orthogonal components, which are the equivalence classes induced by $\pi(b)$, and which we denote by $\pi_1(b), ..., \pi_{k_b}(b)$. We require that blobs in different orthogonal components of b are disjoint. We also require that for any pair of blobs x and y that satisfy $x \notin \rho^*(y)$, $y \notin \rho^*(x)$, and $\rho^*(x) \cap \rho^*(y) \neq \emptyset$, the blobs in $\rho^*(x) \cap \rho^*(y)$ are contained in exactly one orthogonal component of x and one orthogonal component of y.

Now, the hierarchy of blobs can be captured by a strictly alternating AND/OR DAG, whose AND-vertices are the blobs in B. The successors of and AND-vertex b are OR-vertices corresponding to b's orthogonal components. The successors of an OR-vertex v that corresponds to the equivalence class $\pi_i(b)$ are AND-vertices, one for each element in $\pi_i(b)$. This DAG is the AND/OR *containment DAG* of the higraph.

In Fig. 1 we have $B = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o\}$, $E = \{(i, h), (b, j), (l, c)\}$, $\rho(a) = \{b, c, h, j\}$, $\rho(b) = \{d, e\}$, $\rho(c) = \{e, f\}$, $\rho(g) = \{h, i\}$, $\rho(j) = \{k, l, m, n, o\}$ and $\rho(d) = \rho(e) = \rho(f) = \rho(h) = \rho(i) = \rho(k) = \rho(l) = \rho(m) = \rho(n) = \rho(o) = \emptyset$. For each $b \notin \{a, j\}$, $\pi_1(b) = \rho(b)$ and $\pi_i(b) = \emptyset$ whenever $i \neq 1$, since these blobs do not contain more than one orthogonal component. In contrast, for a and j we have $\pi_1(a) = \{b, c, h\}$, $\pi_2(a) = \{j\}$, $\pi_1(j) = \{k, l, m\}$, and $\pi_2(j) = \{n, o\}$. Fig. 2 shows the containment DAG of this higraph, and Fig. 3 shows its AND/OR containment DAG.

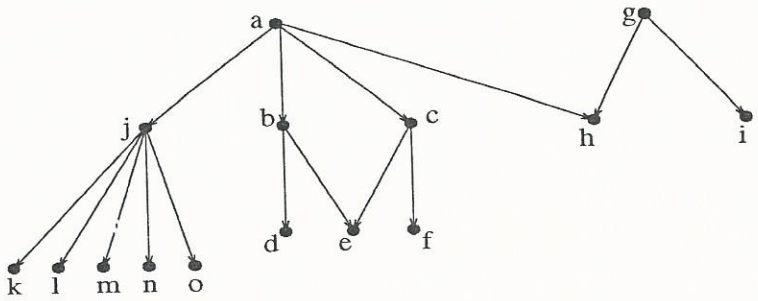

Figure 2: The containment DAG of the higraph of Fig. 1

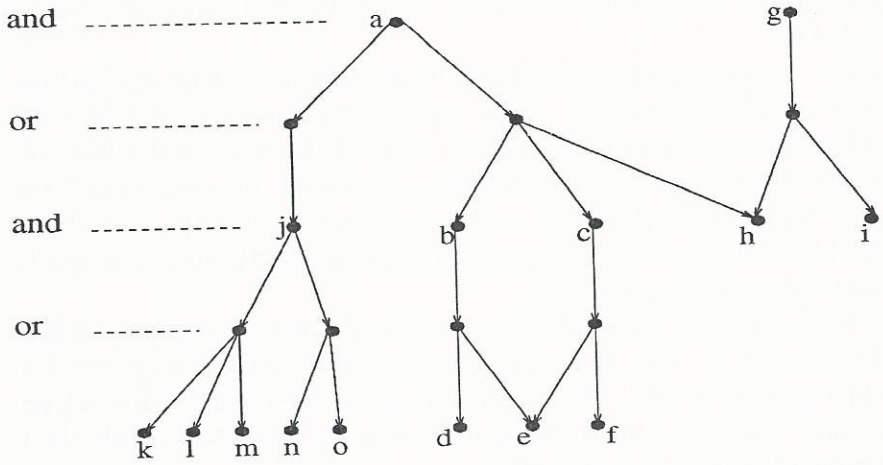

Figure 3: The and/or containment DAG of the higraph of Fig. 1

Definition 2 *A blob x is a* superblob *of y if $y \in \rho^*(x)$. A blob b is* elementary, *or* atomic, *if $\rho(b) = \emptyset$. We say that b is a* descendant *of b', or that b' is an* ancestor *of b, if either $b = b'$ or b is contained in b'. If b is a descendant of b' but $b \neq b'$, we say that b is a* proper descendant *of b', or that b' is a* proper ancestor *of b. Blob b is* directly contained *in blob b' if b is a proper descendant of b' and there is no blob that is a proper descendant of b' and a proper ancestor of b. We use $anc(b)$ to denote the set of b's ancestors, and we write $anc(a, b)$ if $a \in anc(b)$ or $b \in anc(a)$. For an edge $e = (b_1, b_2)$ we denote the* source blob b_1 *of e by e_s and the target blob b_2 by e_t. We also let $SUP(a, b) = \{x : x$ is a superblob of both a and $b\}$.*

For example, in Fig. 1, e is a proper descendant of a, e is a proper descendant of b, and e is directly contained in b but not in a. Also, $anc(e) = \{a, b, c, e\}$.

Definition 3 *Two blobs x and y are* orthogonal, *denoted $orth(x, y)$, if $x = y$ or if there exist blobs b,c and d such that b contains r orthogonal components and there exist $1 \leq i, j \leq r$, with $i \neq j$, such that $c \in \pi_i(b)$, $x \in \rho^*(c)$, $d \in \pi_j(b)$ and $y \in \rho^*(d)$. A set of blobs X is orthogonal if any two elements therein are orthogonal. An orthogonal set of blobs X is* maximal orthogonal *if for each $y \in B$, if $y \notin X$ then $X \cup \{y\}$ is not orthogonal.*

In Fig. 1, for example, $orth(b, j), orth(c, k)$ and $orth(e, n)$ hold, and $\{d, l, n\}$ is an orthogonal set.

2.2 Configurations

The central notion needed for defining a useful set-theoretic semantics for higraphs is that of a legal configuration of blobs, or simply a *configuration*. Before defining configurations, however, we should comment on blob intersection. The relationship between orthogonal blobs is intended to be an AND, depicting Cartesian product, and that of disjoint blobs is a XOR. Similarly, as we shall see, ancestorship gives rise to a form of AND too. The question arises as to the appropriate relationship between intersecting blobs, and we have chosen to take the XOR approach here: blob a intersecting blob b means that a or b will not be allowed to be both part of the same configuration. This is in contrast to [6], where an OR interpretation is adopted for the intersection of states in statecharts; the reader can find justification for that decision there.

Definition 4 *A set of blobs C is a* configuration *if for any non-elementary $b \in C$ containing r orthogonal components, $\mid \pi_i(b) \cap C \mid = 1$ holds for each $1 \leq i \leq r$, and in addition for any $b \in C$, if $\rho^{-1}(b) \neq \emptyset$ then $\mid \rho^{-1}(b) \cap C \mid = 1$.*

A configuration, then, is the set of blobs corresponding to the vertices constituting a legal trace of the AND/OR containment DAG of the higraph. This definition creates two symmetric ways of capturing XOR — by nonintersecting subblobs or by intersecting suberblobs, corresponding to a top-down and bottom-up branching in the DAG, respectively: If a and b are non-intersecting and are contained in c, and c contains no other blobs, then c is the XOR of a and b; and dually, if a and b intersect and their intersection contains blob c and none other, then c is also the XOR of a and b.

In way of justifying this definition, we can consider some of the alternatives. In [9], a legal configuration of a non-overlapping statechart is defined as a maximal orthogonal set of elementary elements (i.e., atomic ones, with no descendants). We could have done the same

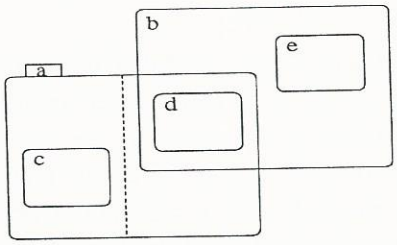

Figure 4: The set $\{d\}$ is not maximal

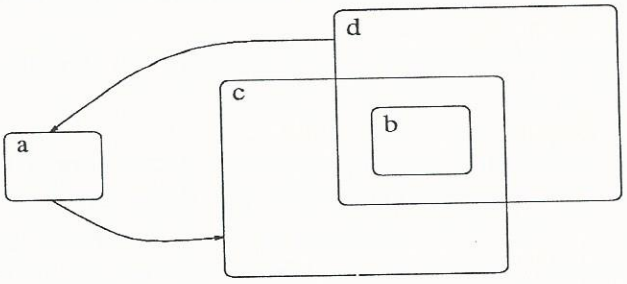

Figure 5: Does this higraph contain a cycle?

here, defining a configuration as a maximal orthogonal set of elementary blobs. However, as indicated in [6], this definition is not suitable for statecharts with overlapping states, and for similar reasons it is not suitable for higraphs with intersections. To illustrate one of the problems with this definition, it seems natural that blob $\{d\}$ in Fig. 4 should be a legal configuration of elementary blobs, since it is contained in b, and b consists of a single orthogonal component. However, $\{d\}$ is not maximal. We should remark that the definition of a configuration in the semantics of statecharts as implemented in the STATEMATE system is consistent with the one we adopt here; see [7].

Another possibility is to adopt our definition above but to have configurations consist of elementary blobs only; this way a configuration would correspond to the set of *leaves*, not vertices, in a legal trace of the AND/OR containment DAG. The main problem with this definition is that specifying only elementary blobs gives rise to ambiguities in the interpretation. For example, consider Fig. 5. Using the terminology of statecharts for convenience, we may say that being in configuration $\{b\}$ does not determine which of c or d we are in. This problem becomes more acute when handling edges, since we may ask the following kinds of questions: (i) Does the higraph of Fig. 5 contain a cycle, although there is no edge from a to d or to a blob contained in it, and there is no edge from c or from a blob it contains to a? (ii) Is a related to $\{e, f\}$ by the edge in Fig. 6, although $\{e, f\}$ is not an element of b, the target blob of the edge? Thus, it seems that a configuration must contain some appropriate 'upward closure' of its elementary blobs.

To substantiate our earlier remarks on succinctness, consider first Fig. 7, with n orthogonal components. According to our definition, the $2n + 1$ blobs yield 2^n configurations. Also, for higraphs with intersection but with no Cartesian products (i.e., no orthogonality) consider Fig. 8 with a total of n blobs, $\sqrt{n}$ of which are the external, mutually intersecting ones, another $\sqrt{n}$ are inside these and also mutually intersecting, and so on. The final $\sqrt{n}$ blobs are in the middle and are mutually exclusive. Thus, n blobs yield $\sqrt{n}^{\sqrt{n}}$ configurations.

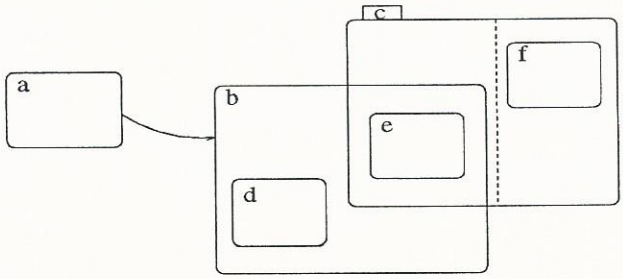

Figure 6: What does the edge (a, b) mean?

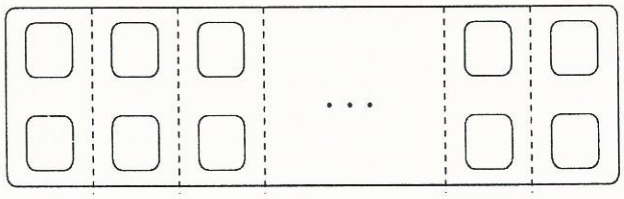

Figure 7: Succinctness of higraphs with Cartesian products but no intersections

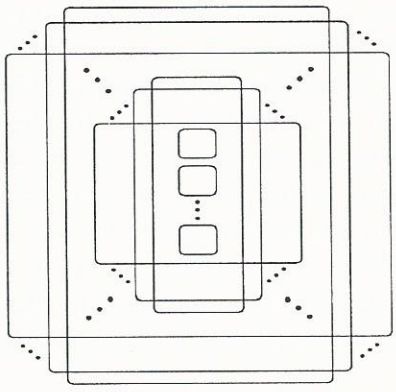

Figure 8: Succinctness of higraphs with intersections but no Cartesian products

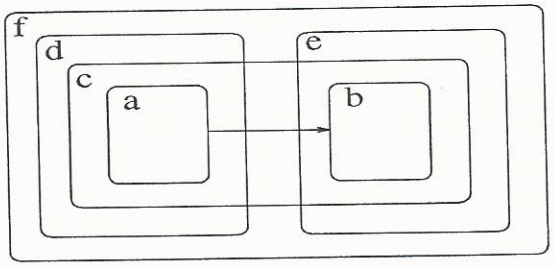

Figure 9: The pros and cons in edge semantics

2.3 Edge semantics

Higraph edges connect blobs. To define their semantics, we interpret them as constituting a binary relation on the induced graph, i.e., as a binary relation on configurations. We shall often borrow from statecharts the term *transition* to denote the effect of an edge in the higraph on the corresponding pair of configurations in the induced graph.[1]

Definition 5 *Given an edge $e = (a, b)$ in a higraph H, if C_1 and C_2 are configurations, with $a \in C_1$ and $b \in C_2$, then e* represents a transition between C_1 and C_2 *in H's induced graph.*

This is a simple 'all-to-all' interpretation. Its main advantage is simplicity, and graphs induced by undirected higraphs are also undirected. Its main disadvantage is that it can sometimes be counter-intuitive. Fig. 9 illustrates this: We do not expect configuration $\{b, e, f\}$ to be a successor of $\{a, c, f\}$ by the edge shown, since there is no reason to leave blob c; but by our definition it is such a successor. The definition may lead to somewhat unnatural results when a transition turns out to have a more global effect than anticipated. The problem is that taking an edge between two blobs that share superblobs might not retain the superblobs of the source blob even though they are also superblobs of the target blob.

In an attempt to avoid this difficulty, we could provide a semantics in which the edge (a, b) represents a transition from a configuration C containing a to a configuration C' containing b but such that $x \in C'$ whenever $x \in (C \cap SUP(a, b))$. Such a definition seems to be quite intuitive, since we expect a transition between blobs to cause the common superblobs that appear in a source configuration to appear also in the corresponding target configurations. According to this semantics, each blob in the source configuration that can legally appear in the same configuration as the target blob will appear in every target configuration. With this semantics in mind, the only successor of $\{a, c, f\}$ by the edge (a, b) in Fig. 9 is $\{b, c, f\}$, since c and f are both superblobs of a and b, and hence have to appear in each target configuration. On the other hand, if we were to add an edge (b, a) to the figure, then $\{a, c, f\}$ would be a successor of $\{b, e, f\}$. This example, under this alternative semantics, shows that the graph induced by an *un*directed higraph may be directed, which is quite disturbing. In fact, such a semantics causes higraphs that seem to be undirected to distinguish between the 'two directions' of an undirected edge. Choosing a semantics that does not preserve the undirectness of the higraph causes many of the properties defined for undirected graphs to be irrelevant to the case of higraphs.

[1] We assume that an edge is specified simply as a directed connection between one blob to another, without indicating which of the superblobs we insist on 'entering' or 'leaving'. These richer kinds of edges were adopted in the overlapping statecharts of [6].

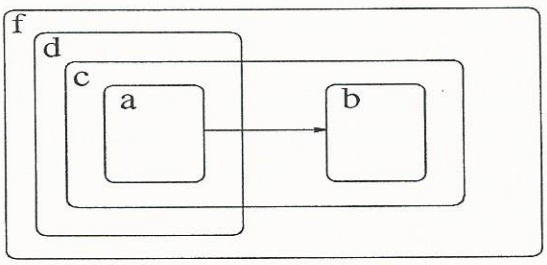

Figure 10: Advantages and disadvantages of the edge semantics

We might try to alleviate this problem by providing a semantics in which the edge (a, b) represents a transition from a configuration C containing a to a configuration C' containing b if for every $x \in SUP(a,b) \cap (C \cup C')$ we have $(\rho^{-1})^*(x) \cap C = (\rho^{-1})^*(x) \cap C'$. Here, C' is a successor of C by edge (a, b) if and only if according to the previous semantics the undirected edge (a, b) causes C' to be a successor of C and C to be a successor of C'. Thus, we ensure that the graph induced by an undirected higraph is also undirected. The only blobs that may be contained in only one of the source and target configurations are the ones contained in the 'smallest' superblob that appears in both configurations. For example, in Fig. 9, the the edge (a, b) defines the following pairs of mutually connected configurations: $\{a, c, f\} \leftrightarrow \{b, c, f\}$ and $\{a, d, f\} \leftrightarrow \{b, e, f\}$. This proposal suffers from two main problems. First, the set of successors is often not the one we would intuitively expect. Second, and more important, in some cases the set of successors of a configuration containing the source blob may be empty. In Fig. 10, the configuration $\{f, d, a\}$ has no successor by the edge (a, b).

We have thus decided to stick to the simple all-to-all semantics for edges.

3 The shortest path problem

In this section we work with weighted higraphs, for which there is a weight (or length) $l(e)$ attached to each edge e. Paths are taken to be paths of transitions between configurations, as in Def. 5.

Definition 6 *The* shortest path *between two blobs s and t in a higraph is defined as the shortest path in the induced graph between any configuration containing the source blob s and any configuration containing the target blob t.*

For convenience, we will denote by $dist_s(b)$ (or by $dist(b)$ if s is clear from the context) the length of the shortest path between the source blob s and a blob b.

We present an algorithm for higraphs with unit lengths, i.e., with $l(e) = 1$ for all $e \in E$, and two algorithms for higraphs with positive lengths. Under these assumptions, just like with ordinary graphs, the algorithm for finding the shortest path between two given blobs and the algorithm for finding the shortest paths from a given source blob to all others, are essentially identical.

3.1 Unit-length edges

The algorithm SP1 we now present is an extension of Moore and Dijkstra's algorithm for ordinary directed graphs with unit-length edges [1].

The idea is to assign labels to blobs, with b's label, $d(b)$, capturing the length of the shortest path found so far from the source blob s to b. First, we assign label 0 to all the blobs that can be in the same configuration with s. We then iterate, so that in the i'th iteration the algorithm assigns label i to all as of yet unlabeled blobs that are contained in a configuration that can be reached by traversing an edge from some configuration containing a blob labeled $i-1$. In order to improve the algorithm we employ a variable L, whose value is the set of blobs that are not end-points of any of the edges already handled.

Algorithm SP1

```
1.   procedure d(s,t)
2.      L ← B − {s}
3.      for each b ∈ B do  d(b) ← ∞  end
4.      for each b ∈ B such that (anc(b,s) or orth(b,s)) do  d(b) ← 0  end
5.      i ← 0
6.      while d(t) = ∞ and ∃b ∈ B such that d(b) = i do
7.         for each b ∈ B such that d(b) = i do  LabelSuccessors(b)  end
8.         i ← i + 1
9.      end
10.     return (d(t))
11.  end

12.   procedure LabelSuccessors(b)
13.      for each e ∈ E such that (e_s = b and e_t ∈ L) do
14.         L ← L − {e_t}
15.         for each w ∈ B such that (d(w) = ∞ and (anc(e_t,w) or orth(e_t,w))) do
16.            d(w) ← i + 1
17.         end
18.      end
19.   end
```

SP1 terminates when the target blob has a label other than ∞, or when there is an iteration of the loop in lines 6–9 in which no blob is assigned a label. Since the label on a blob can be changed at most once, and since there are finitely many blobs, SP1 always terminates.

To show that SP1 really computes the distance between s and t, we show, by induction on $d(b)$, that upon termination $d(b) = dist_s(b)$ for each $b \in B$. For $b = t$, this is what we want.

Clearly, $d(b)$ is assigned 0 if and only if it gets assigned in line 4, and this happens if and only if b has the property that it can be in the same configuration with s, and hence $dist(b) = 0$. Now, since each blob can be assigned a label other than ∞ at most once, and since labels are assigned to the blobs in non-decreasing order, we may assume by induction that for every $d(b) \leq i$ we have $dist(b) = d(b)$. In iteration $i+1$ of the loop in lines 6–9 we assign label $i+1$ to each blob b with $d(b) = \infty$ and with the property that a configuration containing b can be reached by traversing one edge from a configuration containing a blob c with $d(c) = i$. Thus, each blob b assigned a label in this iteration is at distance 1 from a blob which, by the inductive hypothesis, is at distance i from s. Hence, such a b is at distance

$i+1$ from s. On the other hand, any blob that is at distance 1 from a blob that is at distance i from s, indeed gets assigned the label $i+1$.

To analyze the time complexity of SP1, let $n = |B|$ and $m = |E|$. Since the algorithm changes the label of each blob at most once, it can enter the loop in lines 6–9 at most once for each blob. Thus, LabelSuccessors is executed at most once for each blob, and hence is executed at most n times altogether. The main loop in this procedure (lines 13–18) is reached at most once for each edge (b, v), but the assignment in line 14 causes it to be executed at most once for each blob $v \in L$, and hence it is carried out a total of at most n times. Consequently, the algorithm's complexity depends on the difficulty of evaluating the test in line 15, which is dominated by computing the number of blobs that can appear with a given blob in the same configuration; namely, the number of blobs that are ancestors or descendants of a given blob or are orthogonal to it. In the worst case this takes time $O(n)$ for each blob, by searching the DAG of blobs (see Section 3.2.1). Thus, the worst case time complexity of SP1 is $O(n^2)$.

In the special case where H happens to be an ordinary directed graph, the complexity $O(m)$, since the test in line 15 takes only constant time. Thus, on graphs algorithm SP1 is comparable to known algorithms [1], though the constants might be larger.

3.2 Positive edge lengths

This section presents two algorithms for finding shortest paths in higraphs with positive edge lengths. Algorithm SP2 is another variant of the Moore and Dijkstra algorithm [1]. Algorithm SP3 is less similar to known algorithms, but for most higraphs it will be more efficient than SP2.

3.2.1 Algorithm SP2

The algorithm performs $|B|$ iterations, at the start of each of which B is partitioned into two subsets, S and T, with $s \in S$. Each blob b is assigned a label $d(b)$ that we shall prove has the following properties: (i) if $b \in S$, then $d(b) = dist(b)$, and (ii) if $b \in T$, then $d(b) = \min_{e \in E, e_s \in S, anc(e_t, b)}(d(e_s) + l(e))$

For $b \in T$, $d(b)$ will be the length of the shortest path from s to b, subject to the condition that all the blobs along that path, with the exception of b, are in S.

Algorithm SP2

1. $T \leftarrow B$
2. **for each** $b \in T$ **do** $d(b) \leftarrow \infty$ **end**
3. **for each** $b \in T$ **such that** $anc(a, b)$ or $orth(a, b)$ **do** $d(b) \leftarrow 0$ **end**
4. **while** $T \neq \emptyset$ **do**
5. find $b \in T$ such that $d(b) = \min_{b' \in T} d(b')$
6. $T \leftarrow T - \{b\}$
7. **for each** $e \in E$ **such that** $e_s = b$ **do**
8. **for each** $b' \in T$ **such that** $anc(e_t, b')$ or $orth(e_t, b')$ **do**
9. $d(b') \leftarrow \min(d(b'), d(e_s) + l(e))$
10. **end**
11. **end**
12. **end**

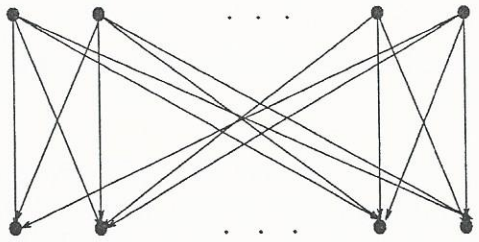

Figure 11: The DAG for a $\Theta(n)$-blob higraph with $\Theta(n)$ descendants or ancestors

The proof of correctness is similar to that of the Moore and Dijkstra algorithm [1]. In each iteration of the loop in lines 4–12, one blob is deleted from T, and hence the loop is executed $|B|$ times, and algorithm SP2 terminates.

We shall prove that whenever algorithm SP2 reaches line 5, any blob b that satisfies $d(b) = \min_{b' \in T} d(b')$ must also satisfy $d(b) = d(b)$. Clearly, there exists a path of length $d(b)$ from s to b. To show that there is no shorter path, consider any path μ from s to b, and divide it into two parts: μ_1, going from s to c, and μ_2, going from d to b. For c and d we thus have either $anc(c,d)$ or $orth(c,d)$. This division is such that one of c or d is the first blob of T to occur along μ. If $c \in T$, then $|\mu_1| \geq d(c)$ (since $d(c)$ is the length of the shortest path to c that passes only through blobs in S) and $d(c) \geq d(b)$ (since b is the next element in T to be added to S). Thus, $|\mu_1| \geq d(b)$. If $c \in S$, then $d \in T$, and hence $|\mu_1| \geq d(d) \geq d(b)$. Since $|\mu_2| \geq 0$, in both cases $|\mu| \geq d(b)$. This concludes the correctness proof.

For the complexity analysis, letting $n = |B|$ and $m = |E|$, as before, we shall see that the time depends to some extent on the data structure used. Clearly, the complexity of SP2 depends on the main loop in lines 4–12. Each iteration in line 5 requires $|T|$ comparisons, so that the total number of comparisons will be $n+(n-1)+...+1 = (n+1)n/2$. The complexity of the loop in lines 7–11 depends on the number of operations required to find, for a given b, all the blobs $b' \in T$ such that $anc(b,b')$ or $orth(b,b')$. Thus, we are faced with the question of how many blobs b' satisfy $anc(b,b')$ or $orth(b,b')$? Consider the higraph corresponding to the containment DAG of Fig. 11. It has $2n$ blobs, n of which are mutually intersecting, and their intersection contains all the other n blobs, which are disjoint. In each iteration of the loop in lines 4–12, a single blob is deleted from T, and hence the average size of T is n. Each blob b that is deleted from T has n blobs b' that satisfy $anc(b,b')$. Thus, on average, it takes $\Theta(n)$ comparisons to find the blobs b' in T that satisfy $anc(b,b')$. The loop in lines 7–11 is executed once for each edge, and since $\Theta(n)$ blobs b' in T can satisfy $anc(e_t,b')$, the complexity of the internal loop may be $\Omega(n)$ for each of the m edges. The total complexity for this case is thus $\Theta(mn) + \Theta(n^2)$.

When m is not $O(n)$ (it can be $\Theta(n^2)$), the first term is dominant, and to improve the bound we must use an algorithm that does not have to find the set $anc(e_t,b')$ for each edge e, such as algorithm SP3 described below. However, when the second term is dominant, we can use a different data structure for the elements of T to improve the time bound. (So far, we were tacitly assuming that T is maintained simply by its characteristic vector.) If T is represented by a heap, for example, only a constant amount of time will be needed to find $\min_{b' \in T} d(b')$, and another $O(\log |T|)$ to find and update the value of $d(b')$ in line 9 (if needed).

If the higraph happens to be an ordinary directed graph, algorithm SP2 behaves like

that of Moore and Dijkstra, and has similar complexity. In each execution of line 8 it uses constant time to discover that the only blob b' satisfying $anc(b, b')$ is b. Thus, the overall complexity depends on the time to find $\min_{b' \in T} d(b')$ and to find and update $d(b')$. This, in turn, depends on the data structure being used.

In summary, for dense graphs (i.e., with $m = \Theta(n^2)$) the complexity of SP2 is $\Theta(n^2)$, and for sparse graphs, using heaps, it reduces to $\Theta(m \log n)$.

3.2.2 Algorithm SP3

Algorithm SP3 finds the shortest paths from a given blob to all others in a higraph with positive edge lengths, and its time bounds are better than those of SP2. Like SP2, algorithm SP3 performs $|B|$ iterations, at the start of each of which B is partitioned into S and T. In fact, in each iteration, the blob b that satisfies $d(b) = \min_{b' \in T} d(b')$ is deleted from T and added to S. Here, each blob b is assigned two labels, $d(b)$ and $e(b)$. We shall prove that if $b \in S$ then $d(b) = dist(b)$, and if $b \in T$ then $d(b) = \min_{e \in E, e_s \in S, e_t = b}(d(e_s) + l(e))$. Also, $e(b)$ is the length of the shortest path from s to b, subject to the conditions that all the blobs along the path, other than b, are in S, and that the target blob of the last edge is b. Thus, if $e(b) = x$, each b' satisfying $anc(b, b')$ also satisfies $d(b') \leq x$.

Algorithm SP3

1. $T \leftarrow B$
2. **for each** $b \in T$ **do** $d(b), e(b) \leftarrow \infty$ **end**
3. $d(s), e(s) \leftarrow 0$
4. **while** $T \neq \emptyset$ **do**
5. find $b \in T$ such that $d(b) = \min_{b' \in T} d(b')$
6. **for each** $b' \in T$ **such that** $anc(b, b')$ or $orth(b, b')$ **do** $d(b') \leftarrow \min(d(b'), e(b))$ **end**
7. $T \leftarrow T - \{b\}$
8. **for each** $e \in E$ **such that** $e_s = b$ **do**
9. $d(e_t) \leftarrow \min(d(e_t), d(e_s) + l(e))$
10. $e(e_t) \leftarrow \min(e(e_t), d(e_s) + l(e))$
11. **end**
12. **end**

Termination is established as in SP2. Denote by $d_{SP2}(b)$ and $d_{SP3}(b)$ the labels that algorithm SP2 and algorithm SP3 assign to b, respectively. The correctness proof uses the fact that, in both algorithms, any value assigned to $d(b)$ during the execution of the algorithm is an upper bound on the value of $d(b)$ upon termination. We prove that for each b, $d_{SP2}(b) = d_{SP3}(b)$.

First, we show that if $d_{SP2}(b) = i$ then $d_{SP3}(b) \leq i$. The claim is trivially true for any b with $d(b) = 0$. Now, assume that each blob b with $d_{SP2}(b) < i$ satisfies $d_{SP3}(b) \leq d_{SP2}(b)$. For b with $d_{SP2}(b) = i$ we show that $d_{SP3}(b) \leq i$. Algorithm SP2 assigns a value to d in line 9, from which it is clear that there is a blob a and an edge e such that $d_{SP2}(a) = j < i$, $e_s = a$, $l(e) = i - j$, and $b \in anc(e_t)$ or $orth(b, e_t)$. By the inductive hypothesis, $d_{SP3}(a) \leq d_{SP2}(a)$. Now, consider the values of the variables of algorithm SP3 at the iteration in which a is chosen to be the blob satisfying $d(a) = \min_{b' \in T} d(b')$. If $b \notin T$, then $b \in S$, and we have $d_{SP3}(b) \leq j < i = d_{SP2}(b)$. If $b \in T$, then the assignments in lines 9–10 yield $e(e_t), d(e_t) \leq j + (i - j) = i$, and hence when SP3 deletes e_t from T the assignment in line 6

yields $d_{SP3}(b) \leq e(e_t) \leq i$.

We now show that if $d_{SP3}(b) = i$ then $d_{SP2}(b) \leq i$. As before, the claim is true for each b with $d(b) = 0$. Assume that each blob b with $d_{SP3}(b) < i$ satisfies $d_{SP2}(b) \leq d_{SP3}(b)$. Let b satisfy $d_{SP3}(b) = i$, and we show that $d_{SP2}(b) \leq i$. Algorithm SP3 assigns a value to d in lines 6 and 9. If $d_{SP3}(b)$ gets its value in line 9, then $d_{SP3}(b) = d(e_s) + l(e)$, for some edge e satisfying $b = e_t$ and $d(e_s) \geq d_{SP3}(e_s)$ (since the values of the label d cannot increase). Since $l(e)$ is positive at the instant of the assignment, $d(e_s) = j < i$, and since the values of the label d cannot increase, $d_{SP3}(e_s) \leq j < i$. By the inductive hypothesis, $d_{SP2}(e_s) \leq d_{SP3}(e_s)$. Now, executing line 9 of algorithm SP2 in the iteration in which the algorithm deletes e_s from T, and assuming that $b \in T$ in this iteration, yields $d_{SP2}(b) \leq d_{SP2}(e_s) + l(e) \leq i$. If $d_{SP3}(b)$ gets its value in line 6, then that value is the one $e(c)$ has at the moment (c is some blob satisfying $anc(b, c)$ or $orth(b, c)$). The algorithm assigns values to $e(c)$ only in line 10, so that there is an edge e with $e_t = c$ and $e(c) \leq d(e_s) + l(e)$. Since $l(e)$ is positive, $d_{SP3}(e_s) \leq j < i$. By the inductive hypothesis, $d_{SP2}(e_s) \leq d_{SP3}(e_s)$. Executing line 9 of algorithm SP2 in the iteration in which the algorithm deletes e_s from T, and assuming that $b \in T$ in this iteration, yields $d_{SP2}(b) \leq d_{SP2}(e_s) + l(e) \leq i$. This completes the correctness proof.

To assess the time complexity, let $n = |B|$ and $m = |E|$, as usual. Again, the complexity of SP3 depends on the main loop in lines 4–12. The iterations in line 5 require $(n + 1)n/2$ comparisons. Just as in SP2, the complexity of line 6 depends on finding the blobs b' such that $anc(b, b')$ or $orth(b, b')$, and the the total time for line 6 may be $\Omega(n^2)$. Since the loop in lines 8–11 runs in time $\Theta(m)$, the overall complexity of the algorithm is $\Theta(n^2)$.

Here too, we can use a heap for sparse higraphs, resulting in a $\Theta(m \log n)$ bound. Also, for the special case of a directed graph, algorithm SP3 behaves like that of Moore and Dijkstra, and has similar complexity.

4 Connectivity and distance-ambiguity

We first define connectivity between blobs, capturing the fact that in many applications the meaning of an edge incident upon a high-level blob is that it applies to all subblobs. Connectivity is, in a way, dual to the notion of a transition (compare with Def. 5):

Definition 7 *The edge* (a, b) connects *blobs* c *and* d *if* $a \in anc(c)$ *and* $b \in anc(d)$.

Given this definition, the labels on edges in a higraph cannot be viewed naively as lengths, or distances. In Fig. 12, for example, the edge from c to itself implies that the distance $d(h, i)$ is 3, whereas there is a direct edge from h to i labeled 2. To overcome such inconsistencies, we adopt a general priority principle that has inner edges *override* higher-level ones, as is done in [10, 11] (and also in early versions of the STATEMATE system [8]). To illustrate priorities, we interpret Fig. 12 as follows. The blobs of c are at distance 3 from each other, except that $d(h, i) = 2$. The distance from the blobs of c to those of a is 5, but the distance from a to the blobs inside c is 4. Also, $d(d, f) = 3$ since the edge (a, b) is of length 3, and d and f are contained in a and b, respectively. Now, $d(e, f) = 6$, not 3, since (e, f) overrides (a, b). Finally, $d(e, j)$ is not well defined, since the length of (a, c) is 4, while the length of (b, c) is 7, and e is contained in both a and b; we thus have ambiguity.[2]

[2] We could make an arbitrary decision to resolve ambiguity, such as to determine length by the shortest edge, as was done, in fact, in the semantics of Section 3.2. Here we take the approach that users want to be notified of ambiguity.

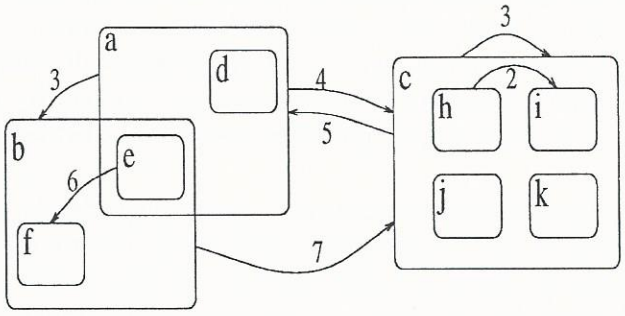

Figure 12: Distance-ambiguity and priorities

4.1 Distance tables

In this section we suggest an efficient algorithm for detecting distance-ambiguity and for computing distances whenever they are unambiguous. We restrict ourselves to higraphs without Cartesian products, since otherwise it is difficult to make distances well-defined. In the presence of Cartesian products, configurations may have more than one elementary blob, and requiring the distance between elementary blobs to be uniquely defined implies that any pair of elementary blobs that can appear in the same configuration must be at the same distance from any other blob. In other words, if (a_1, b_1) and (a_2, b_2) are two edges with different lengths, and $orth(a_1, a_2)$ and $orth(b_1, b_2)$, then the two edges contradict each other. Besides, edges between orthogonal components have to be of length 0, since it makes sense to require that the distance between two blobs that can appear in the same configuration should be 0.

Definition 8 *An edge e* strictly overrides *e', denoted $e \ll e'$, if the source and target of e are respective ancestors of those of e', but $e \neq e'$. An edge e* overrides *e', denoted $e \lessdot e'$, if $e \ll e'$ or $e = e'$. (Thus, $e \lessdot e'$ iff $e_s \in anc(e'_s)$ and $e_t \in anc(e'_t)$.) If E and F are sets of edges, with $E \subseteq F$, we say that E* overrides *F, written $E \ll F$, if for each $f \in (F - E)$ there exists $e \in E$ such that $e \ll f$.*

Throughout this section, we shall denote by B' the set of elementary blobs of a higraph with blob set B. We can now define the distance function d, which is inspired by the semantics for the higraph-based language for security systems of [10, 11].

Definition 9 *Let $Gov(a, b)$ to be the set of edges 'relevant' to blobs a and b, defined as follows:*

$$Gov(a, b) = \{e \in E \mid e_s \in anc(a) \text{ and } e_t \in anc(b)\}.$$

The distance function, *or* distance table, *$d : B' \times B' \to R \cup \{\infty\} \cup \{\text{'amb'}\}$ is defined as follows:*

$$d(a, b) = \begin{cases} 0 & a = b \\ \infty & a \neq b \text{ and } Gov(a, b) = \emptyset \\ x & a \neq b,\ Gov(a, b) \neq \emptyset \text{ and } \forall e \in Gov(a, b)\ \exists e' \lessdot e\ :\ l(e') = x \\ \text{'amb'} & \text{otherwise} \end{cases}$$

We thus have 'amb' in a and b's entry in the distance table whenever the distance between them is ambiguous. The purpose of this section is to compute the distance table.

The efficiency of an algorithm for computing the distance table obviously depends on the representation of the higraph. We assume a representation in which each blob entry contains pointers to all its proper descendants and all its proper ancestors, as well as a list of the edges incident to it.

As usual, we denote by n and m the number of blobs and edges, respectively. Both m and the total size of the containment lists may reach $O(n^2)$ (the latter will be $O(n)$ when there are no intersections). Computing the distance table will take $\Omega(n^2)$, since it is of size $O(n^2)$. Even the ambiguity problem requires time $\Omega(n^2)$ in the worst case. Hence, the best time possible for any algorithm is $O(n^2)$.

4.2 A naive algorithm

We now present algorithm UL1, which is a straightforward implementation of the definition of d. For each pair (a, b), it computes the set $Gov(a, b)$, and then checks whether $Gov(a, b) = \emptyset$. If this is the case, $d(a, b)$ is assigned ∞; otherwise, the algorithm checks whether there is a subset S of $Gov(a, b)$ having all edges with the same length, such that each edge in $Gov(a, b) - S$ is overridden by an edge in S. If such a subset exists, $d(a, b)$ is set to the common length of the edges in S, otherwise, $d(a, b)$ is 'amb'.

Central to the definition of $Gov(a, b)$ and the overrides relation on edges is ancestor (or containment) relation on blobs. When there are no intersections in the higraph, the containment DAG is a tree, and it is then possible to spend $O(n)$ preprocessing time on a preorder traversal, such that ancestorship can be decided in constant time using only $O(nlogn)$ space. This technique does not generalize to arbitrary DAG's. To decide if $b' \in anc(b)$ in an arbitrary DAG, we carry out a depth-first search upwards from b. This search will run in time roughly proportional to the depth of the DAG, as long as only a constant number of vertices have more than one parent.

We associate a level, $level(b)$, with a blob b, corresponding to its distance from the bottom of the DAG. Blobs corresponding to leaves have level 0, and the level of an internal blob is one plus the maximum level of the blobs it contains. Computing node levels takes only $O(n)$ preprocessing time. When deciding if $b' \in anc(b)$, we stop a particular branch of the search if the node we are visiting has a level greater than $level(b')$.

We use the variable Gov to compute $Gov(a, b)$).

Algorithm UL1:

```
1.    procedure d(a,b)
2       Gov ← ∅
3       for each e ∈ E do
4          if (e_s ∈ anc(a)) ∧ (e_t ∈ anc(b))  then Gov ← Gov ∪ {e}
5       end
6       for each e ∈ Gov do
7          for each f ∈ Gov − {e} do
8             if (e ≪ f)      then Gov ← Gov − {f}
9          end
10      end
```

```
11        if Gov = ∅  then return (∞)
12        else do
13          chose e ∈ Gov
14          if e' ∈ Gov ⇒ l(e) = l(e')  then return (l(e))
15          else return ('amb')
16        end
17      end
```

For the purpose of analyzing this algorithm, assume we implement an ancestorship test requiring $O(t)$ time per query. The computation of the set Gov, which may be of size $O(m)$, thus requires $O(mt)$ time. The loop in lines 6–10 to find the edges in Gov that are not overridden by any other edge therein may require $O(m^2t)$ time. Since there are n blobs, there are $O(n^2)$ distances to compute. We conclude that algorithm UL1 requires $O(n^2m^2t)$ time to generate the distance table. Using the constant time ancestorship test, the total time complexity is $O(n^2m^2)$. Space complexity is $O(n^2)$, assuming we use the constant time ancestorship test.

The main problem with algorithm UL1 is that it repeats a great deal of work. For example, consider the computation of $d(a,b)$ and $d(a',b')$, where a and a' are contained in the same set of blobs α, and b and b' are contained in the same set of blobs β. Consider the set E' of edges connecting blobs that contain α with blobs that contain β. Clearly the edges in E' govern the distances between a and b and between a' and b'. Thus, the work required to find the set E' and to add it to Gov is performed for each pair. An efficient algorithm would eliminate this repetitive work, and would also detect only once those edges in E' that are strictly overridden by other edges in E'.

4.3 A better algorithm

This subsection describes algorithm UL2, which makes one pass instead of two, and computes information not only for the leaves of the containment DAG, but for each pair of blobs. Moreover, computed information can be used more than once. The algorithm computes the set $Gov(a,b)$ for each pair of blobs a and b, by performing a depth-first search on the containment DAG.

Definition 10 *Let $A' \subseteq E$ be a set of edges. The* minimal edges *of A', written $min(A')$, are those that are not strictly overridden by any other edge in A':*

$$min(A') = \{a' \in A' : \forall a \in A', a \lll a' \Longrightarrow a = a'\}.$$

The only edges that matter in determining $d(a,b)$ are those in the set $Gov(a,b)$, which we shall abbreviate to A'. The distance between a and b is well defined if all the edges in $min(A')$ have the same length. In order to simplify the definitions and the algorithm, we assume that all blobs in the higraph are enclosed in a common 'super-blob' $\hat{b}$.

We denote by $danc(b)$ the set $\{b' \in B \mid b' \text{ directly contains } b\}$, and by $ddes(b)$ the set $\{b' \in B \mid b' \text{ is directly contained in } b\}$.

Definition 11 *For each pair of blobs a and b, $W(a,b)$ is defined recursively as follows:*

$$W(a,b) = \begin{cases} \emptyset & \text{if } a = \hat{b} \text{ or } b = \hat{b} \\ \{(a,b)\} & \text{if } (a,b) \in E \\ min[W(danc(a),b) \cup W(a,danc(b))] & \text{otherwise} \end{cases}$$

Also,

$$W(A,b) = \bigcup_{a \in A} W(a,b) \;\; ; \qquad W(a,B) = \bigcup_{b \in B} W(a,b)$$

We will show that $W(a,b) = min(Gov(a,b))$. That is, for each pair of blobs a and b, $W(a,b)$ is the witness set for that pair, and $d(a,b)$ can be computed from $W(a,b)$.

Definition 12 *For any pair of blobs a and b, such that level(a) = level(b) = 0, the generalized distance function, dist(a, b), is defined as follows:*

$$dist(a,b) = \begin{cases} 0 & \text{if } a = b \\ \infty & \text{if } a \neq b \text{ and } W(a,b) = \emptyset \\ x & \text{if } a \neq b \text{ and each } e \in W(a,b) \text{ satisfies } l(e) = x \\ \text{'amb'} & \text{otherwise} \end{cases}$$

Algorithm UL2 iterates first over the source blobs and then over the target blobs. For each source blob b, it computes the *block* of witness sets $Blk(b)$. The block contains the witness sets for b and every other blob c. Our notation treats the block as an array of witness sets indexed by blobs; we write $Blk(b)[c]$ to refer to the witness set $W(b,c)$.

We compute the witness set blocks in a roughly depth-first order, optimizing for the case where the containment DAG is tree-like. However, if a blob is contained in more than one other blob, we must compute the blocks for each of its parents before continuing the downward recursion. The computation for the parents may require the blocks for their parents, and these are computed using recursion, which is stopped when a vertex whose blob has already been computed is encountered.

The algorithm minimizes its memory requirements by classifying each object as *unmarked*, *visited*, or *completed*. Initially, all blobs are unmarked. When we first visit a blob during the search (as a source blob), we allocate space for its block, and mark it as visited. Once we have finished visiting every descendant of that blob, we have no further use for any of the witness sets in the blob's block, and we deallocate the space for the block. We also mark the blob as completed, so that we do not search it again if encountered along some other path.

The main program and procedures ComputeWBlocksBelow and ComputeWBlockFor traverse the containment DAG of the higraph, marking vertices as necessary to the computation of witness set blocks when needed and to deallocate those blocks when they are no longer needed. The procedure FillWBlock(b) computes the witness sets of $Blk(b)$. It iterates over the blobs in such a way that it always computes the witness sets between the parents of blob b and c before it actually computes $W(b,c)$. This is done by iterating over the target blobs in order of decreasing level. For each target blob c, $W(b,c)$ is computed by calling the procedure ComputeW(b,c). If b and c are both elementary blobs, then $d(b,c)$ is assigned a value according to Definition 12.

The procedure ComputeW(b,c) computes and returns the witness set $W(b,c)$. If there are any edges connecting b to c, they determine the witness set. If there are no such edges, the third case of Definition 11 is realized, by considering each of the relevant witness sets in turn, computing the minimal edges in the union of the 'current' minimal edge set *min* and the 'next' relevant parent set. The MinWSet procedure uses a simple quadratic-time algorithm to determine this minimal set of edges.

Algorithm UL2:

```
begin
  allocate space for Blk(b̂)
  for each b ∈ B do
      unmark b
      Blk(b̂)[b] ← ∅
  end
  mark b̂ as visited
  for each b ∈ ddes(b̂) do ComputeWBlocksBelow(b) end
  deallocate space used by Blk(b̂)
end

procedure ComputeWBlocksBelow(b)
  if b is completed  then return
  if b is unmarked  then ComputeWBlockFor(b)
  for each b' ∈ ddes(b) do ComputeWBlocksBelow(b') end
  mark b as completed
  deallocate space used by Blk(b)
end

procedure ComputeWBlockFor(b)
  mark b as visited
  allocate space for Blk(b)
  for each b' ∈ danc(b) do
      if b' is unmarked  then ComputeWBlockFor(b')
  end
  FillWBlock(b)
end

procedure FillWBlock(b)
  for lev from max-blob-level downto 0 do
      for each c ∈ B such that level(c) = lev do
          Blk(b)[c] ← ComputeW(b,c)
          if level(b) = level(c) = 0 then select
              when (b = c)   d(b, c) ← 0
              when (b ≠ c and Blk(b)[c] = ∅)    d(b, c) ← ∞
              when (b ≠ c and for each e ∈ Blk(b)[c], l(e) = dist)    d(b, c) ← dist
              otherwise d(b, c) ← 'amb'
          endselect
      end
  end
end

procedure ComputeW(b,c) returns (WSet)
  if (b, c) ∈ E   then return ({(b, c)})
  else do
      min ← ∅
      for each b' ∈ danc(b)   do min ← MinWSet(min, Blk(b')[c]) end
```

```
      for each c' ∈ danc(c)  do min ← MinWSet(min, Blk(b)[c']) end
      return (min)
  end
end

procedure MinWSet(E1,E2) returns (WSet)
  E1' ← E1
  E2' ← E2
  for each e1 ∈ E1, e2 ∈ E2 do
      if e1 = e2 or e1 ≪ e2  then E2' ← E2' − {e2}
         else  if e2 ≪ e1  then E1' ← E1' − {e1}
  end
  return (E1' ∪ E2')
end
```

To show that UL2 correctly computes the semantic distance $d(a,b)$ between each pair of blobs a and b, we show that $d(a,b) = dist(a,b)$, so that $dist(a,b)$ is the real distance between a and b.

We define $W'(a,b) = min(Gov(a,b))$, and

$$dist'(a,b) = \begin{cases} 0 & \text{if } a = b \\ \infty & \text{if } a \neq b \text{ and } W'(a,b) = \emptyset \\ x & \text{if } a \neq b \text{ and each } e \in W'(a,b) \text{ satisfies } l(e) = x \\ \text{`}amb\text{'} & \text{otherwise} \end{cases}$$

We first show that for each a and b, $d(a,b) = dist'(a,b)$. The proof is then completed by showing that the definition of W implies $W(a,b) = W'(a,b)$, and therefore $d(a,b) = dist(a,b)$. (The proofs of Lemmas 1, 2 and 3 below appear in [10].)

Lemma 1 *Let $A' \subseteq A$ be any set of edges. Then, for all $e' \in A'$, there is $e \in min(A')$ such that $e \ll\!\!= e'$.*

Corollary 1 *$A' \neq \emptyset$ if and only if $min(A') \neq \emptyset$.*

The next lemma shows that the set $min(A')$ is the unique overriding edge set of smallest size.

Lemma 2 *Let $A' \subseteq A$ be any set of edges. Then $min(A') \ll A'$. Moreover, any A'' such that $A'' \ll A'$ satisfies $min(A') \subseteq A''$.*

Proposition 1 *Let a and b be elementary blobs. Then $d(a,b) = dist'(a,b)$.*

Proof: First, it is clear from the definitions of $dist'$ and d that if $a = b$ then $d(a,b) = dist'(a,b) = 0$.

Next, we show that $dist'(a,b) = \infty$ if and only if $d(a,b) = \infty$. Indeed, by definition, $dist'(a,b) = \infty$ implies $W'(a,b) = \emptyset$, i.e., $min(Gov(a,b)) = \emptyset$. Now, $min(Gov(a,b)) = \emptyset$ iff

$Gov(a,b) = \emptyset$; namely, $d(a,b) = \infty$. Conversely, the semantics of d implies that $d(a,b) = \infty$ just when $Gov(a,b) = \emptyset$. In this case, $min(Gov(a,b)) = \emptyset$, and hence $dist'(a,b) = \infty$.

Finally, we show that $dist'(a,b) = x$ if and only if $d(a,b) = x$. For the first direction, let $dist'(a,b) = x$. Also, let $A' = Gov(a,b)$, so that $W'(a,b) = min(A')$. By the definition of $dist'$, $l(e) = x$ for each $e \in W'(a,b)$. By Lemma 2, $W'(a,b) \ll A'$. Hence, for each $e' \in A'$ there exists $e \in W'(a,b)$ such that $e \ll e'$. In particular, for each $e' \in A'$ there exists $e \in A'$ such that $l(e) = x$ and $e \ll e'$. By the definition of d, this is the requirement for $d(a,b) = x$. For the converse direction, assume by way of contradiction that $d(a,b) = x$ and $dist'(a,b) \neq x$. Since we know that $dist'(a,b) \neq \infty$ (recall that $d(a,b) = \infty$ whenever $dist'(a,b) = \infty$), there are two cases to be considered: either $dist'(a,b) = y \neq x$ or $dist'(a,b) =$ *'amb'*. In the former case, the above proof would imply that $dist'(a,b) = y$, which is a contradiction. In the latter case, there must exist an edge $e' \in W'(a,b)$ such that $l(e') = y \neq x$. By the definition of d and the fact that $d(a,b) = x$ there must also exist an edge e such that $l(e) = x$ and $e \ll e'$. However, this is impossible, since $W'(a,b) = min(Gov(a,b))$, so that no edge can strictly override e'. Hence $dist'(a,b) = x$.

We have thus shown that $d(a,b) = \infty$ iff $dist'(a,b) = \infty$, and that $d(a,b) = x$ iff $dist'(a,b) = x$. Since there are only three possible outcomes, we conclude the third of these, namely that also $d(a,b) =$ *'amb'* iff $dist'(a,b) =$ *'amb'*. ■

Lemma 3 *Let $\ll$ be a partial order on S and let $A \subseteq S$. If $A = A_1 \cup A_2 \cup ... \cup A_n$, then*

$$min(A) = min(\bigcup_{i=1}^{n} min(A_i)).$$

Definition 13 *The* depth *of a blob b is given by*

$$depth(b) = 1 + \max_{b' \in Par(b)}(depth(b')),$$

where $depth(\hat{b}) = 0$. For a pair of blobs we sometimes write $depth(a,b)$ for the pair of numbers $(depth(a), depth(b))$. In order to perform induction on the depth of pairs, we define the following partial order on pairs of natural numbers: $(i,j) \prec (i',j')$ if and only if $(i < i' \wedge j \leq j')$ or $(i \leq i' \wedge j < j')$.

Thus, a pair of blobs p is 'deeper' than another pair q if p's elements are pointwise deeper than or equal in depth to those of q, and at least one of the inequalities is strict.

Proposition 2 *For any two blobs a and b, $W(a,b) = W'(a,b)$ (and hence $dist(a,b) = dist'(a,b)$).*

Proof: The proof is by induction on the depth of the pair (a,b). We structure the argument according to three parts of the definition of W in Definition 11:

For the base of the induction one of the components of the depth-pair is 0, which means that $a = \hat{b}$ or $b = \hat{b}$. By the definition of W we have $W(a,b) = \emptyset$. By construction, $Gov(a,b) = \emptyset$, and hence $W'(a,b) = \emptyset$ too.

Now, let the depth of (a,b) be (i,j), with $i,j > 0$. By the induction hypothesis, $W(a',b') = W'(a',b')$ for all pairs of blobs (a',b') with depth (i',j') such that $(i',j') \prec (i,j)$. We have to show that $W(a,b) = W'(a,b)$. There are two cases to consider, corresponding to the second and third lines of the definition of W.

If $(a,b) \in E$, then by definition $W(a,b) = \{(a,b)\}$. We must therefore show that $min(Gov(a,b)) = \{(a,b)\}$. This holds, however, since any edge $e \in (Gov(a,b) - \{(a,b)\})$ is strictly overridden by (a,b). Hence, $\{(a,b)\} \ll Gov(a,b)$, so $min(Gov(a,b)) \subseteq \{(a,b)\}$ by Lemma 3. To see that $\{(a,b)\} \subseteq min(Gov(a,b))$, we simply observe that no edge in $Gov(a,b)$ can strictly override (a,b).

If $(a,b) \notin E$, then by definition we have:

$$W(a,b) \;=\; \min[W(Par(a),b) \cup W(a,Par(b))]. \tag{1}$$

For every edge $e \in Gov(a,b)$ we have $(a \preceq e_s \wedge b \prec e_t) \vee (a \prec e_s \wedge b \preceq e_t)$. Hence:

$$Gov(a,b) = (\bigcup_{b'} Gov(a,b')) \cup (\bigcup_{a'} Gov(a',b)), \tag{2}$$

where $a' \in Par(a)$ and $b' \in Par(b)$. By the inductive hypothesis:

$$\begin{aligned} \forall b' \in Par(b): &\quad W(a,b') = \min(Gov(a,b')) = W'(a,b'), \\ \forall a' \in Par(a): &\quad W(a',b) = \min(Gov(a',b)) = W'(a',b). \end{aligned} \tag{3}$$

Combining equations(1) and (3), we obtain:

$$W(a,b) = \min[(\bigcup_{b'} \min(Gov(a,b'))) \cup (\bigcup_{a'} \min(Gov(a',b)))]$$

By Lemma 3 and equation(2) we obtain:

$$\min(Gov(a,b)) = \min[(\bigcup_{b'} \min(Gov(a,b'))) \cup (\bigcup_{a'} \min(Gov(a',b)))],$$

and thus, $W(a,b) = \min(Gov(a,b)) = W'(a,b)$, as desired. ■

As to the running time of UL2, there are $O(n^2)$ sets of type $Blk(b)[c]$. To compute each entry in the distance table we first search the edge hash table to see if there is an entry for $A(b,c)$. Using dynamic hash tables, we can store and retrieve elements from this table in constant time. If there is no such entry, we must actually compute the entire minimal witness set. The procedure MinWSet($E1, E2$) runs in time $O(|E1| \cdot |E2|)$, where $E1$ and $E2$ are both witness sets. Witness sets can be of size m in the worst case [10], yielding a worst case running time of $O(n^2m^2)$ for the entire algorithm. However, we expect that the average witness set will be much smaller than m.

5 Hamiltonian cycles

A Hamiltonian cycle in a graph is a path through the graph (i.e., along edges), in which each vertex appears exactly once. In this section we suggest some definitions for Hamiltonian cycles in higraphs. There are a number of ways of doing this, and for each we discuss the problem of deciding whether a higraph is Hamiltonian. The analogous problem for graphs is well-known to be NP-complete.

Extending the notion of a cycle to higraphs appears to require that we give up some of the basic properties cycles have in ordinary graphs, such as that in a cycle of more than two vertices no edge appears more than once.

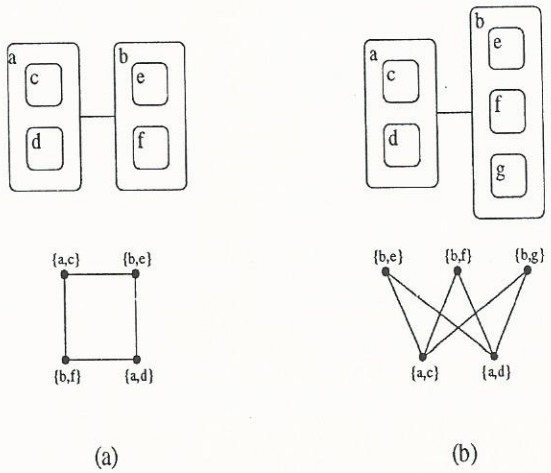

Figure 13: Two higraphs and their induced graphs: (a) is Hamiltonian and (b) is not

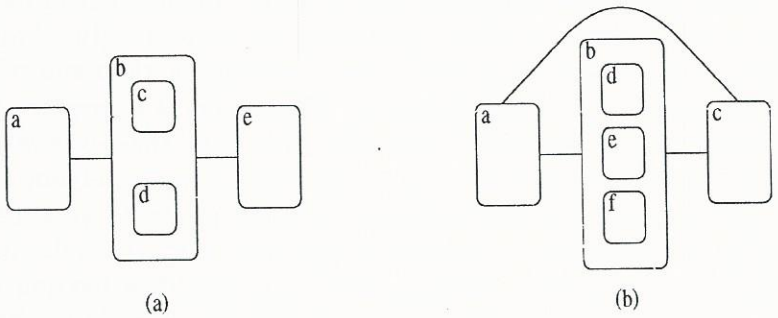

Figure 14: Differentiating cycles of blobs and cycles of configurations

5.1 Hamiltonian cycles of configurations

The first and the most natural definition is to say that a higraph contains a Hamiltonian cycle if and only if there is a Hamiltonian cycle in the induced graph, i.e., the higraph's configurations admit a Hamiltonian cycle. Fig. 13(a) shows a higraph containing a Hamiltonian cycle of configurations, and Fig. 13(b) shows one that does not.

This definition allows a higraph edge to appear more than once in a cycle. Other strange things can also happen; for example, the existence of what might look like a Hamiltonian cycle — a cycle of blobs connected by edges — is not necessarily related to the existence of a Hamiltonian cycle of configurations. In Fig. 14(a) there is no Hamiltonian cycle of the blobs, but there is a Hamiltonian cycle of configurations. Dually, in Fig. 14(b) there is a Hamiltonian cycle of blobs, but there is no Hamiltonian cycle of configurations.

If the higraph contains no intersections and no Cartesian products, the time needed for creating the induced graph is linear in the size of the higraph, and the size of the graph is linear too. Since ordinary graphs are a special case of such higraphs, and since the above definition of a Hamiltonian cycle coincides with the standard definition in the case of ordinary graphs, it follows that deciding whether there is a Hamiltonian cycle of configurations in such a higraph is NP-complete.

The situation is different when Cartesian products or intersections are present, as the size of the induced graph need not be linear in the size of the higraph. Furthermore, the

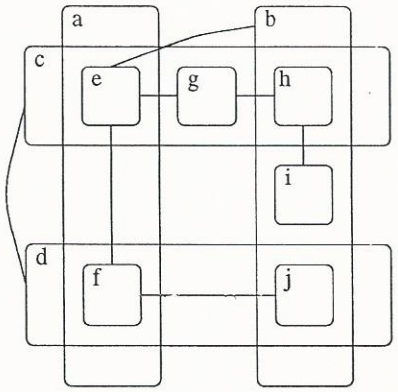

Figure 15: A Hamiltonian higraph with intersections

description of a simple cycle may itself be exponential in the size of the higraph, and thus, the problem of deciding whether a higraph contains a Hamiltonian cycle may not be in NP.

We now want to show that there are Hamiltonian higraphs in which the number of configurations is super-polynomial in the number of blobs, and hence so is the length of any Hamiltonian cycle. To do this, we use a semantics that is less permissible than the 'all-to-all' semantics. In fact, we will prove a stronger claim, by introducing a semantics in which each configuration has exactly one successor by each edge. We show that even with this semantics, there are higraphs that contain Hamiltonian cycles having super-polynomial length. We consider higraphs with intersections and without Cartesian products, in which edges are interpreted according to the semantics presented in Sec. 2.3, with the following addition: A non-elementary blob b may contain a 'default child'; i.e., the identification of a blob $c \in Child(b)$, which has to appear in any configuration containing b that is reached by traversing an edge e with $b \in \rho^*(e_t)$. Also, a blob contained in other blobs (and some of these might be intersecting) can have a similar 'default parent'. By using the possibility of defining such default blobs, one can force any edge in a higraph to admit a maximum of one successor for each configuration.

Fig. 15 is an example of a Hamiltonian higraph with intersections. Its default offspring are defined as follows (in parentheses): $a(e)$, $b(j)$, $c(h)$, $d(j)$. The default parents are defined as follows: $e(a)$, $f(d)$, $g(c)$, $h(b)$, $i(b)$, $j(d)$. The higraph contains the following Hamiltonian cycle: $\{a,e\} \rightarrow \{a,f\} \rightarrow \{d,j\} \rightarrow \{d,f\} \rightarrow \{c,h\} \rightarrow \{b,i\} \rightarrow \{b,h\} \rightarrow \{c,g\} \rightarrow \{c,e\} \rightarrow \{b,j\} \rightarrow \{a,e\}$.

When default blobs are defined, an edge can be used more than once, the number of times depending on the number of ancestors and the number of common ancestors of the edge's endpoints. When two blobs have no common proper ancestor, an edge between them can be used at most twice — once in each direction — since in each use of the edge the target configuration is dictated by the default blobs: Each of the edge's endpoints, when it is the target blob, dictates a unique target configuration that does not depend on the source configuration.

We shall show that it is possible to construct a family of Hamiltonian higraphs with unique successors for each configuration, such that the number of configurations is super-polynomial in the number of blobs. For each $n > 2$ we construct a higraph with n^2 blobs, $n^2 - n + 1$ edges, and n^n configurations, as follows (the result is a variant of the higraph of Fig. 8): Let $\Sigma = \{\sigma_1, \sigma_2, ..., \sigma_n\}$ be an alphabet. The blobs are labeled as follows:

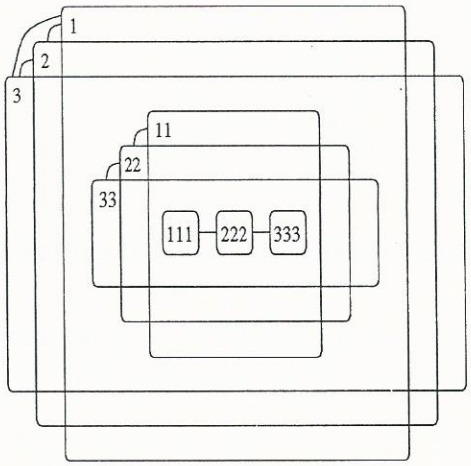

Figure 16: A higraph with 9 blobs and a Hamiltonian cycle of length 27

$\sigma_1, \sigma_2, ..., \sigma_n, \sigma_1\sigma_1, \sigma_2\sigma_2, ..., \sigma_n\sigma_n, ..., \sigma_1^n, \sigma_2^n, ..., \sigma_n^n$. For each $1 \leq j \leq n$, $1 \leq i \leq n-1$, the blob σ_j^i contains the blobs σ_k^{i+1} for $1 \leq k \leq n$. For each σ_j^i, $1 \leq j \leq n$, $2 \leq i \leq n$, the default parent is σ_1^{i-1}, and for each σ_j^i, $1 \leq j \leq n$, $1 \leq i \leq n-1$, the default child is σ_1^{i+1}. The edges are: (σ_1^i, σ_2^i), $(\sigma_2^i, \sigma_3^i), ..., (\sigma_{n-1}^i, \sigma_n^i)$ for all $1 \leq i \leq n$, and (σ_n, σ_1). The higraph contains the following Hamiltonian cycle (see Fig. 16 for an example of the construction for $n = 3$):

$\{\sigma_1, \sigma_1\sigma_1, ..., \sigma_1^n\}$, $\{\sigma_1, \sigma_1\sigma_1, ..., \sigma_1^{n-1}, \sigma_2^n\}$, $\{\sigma_1, \sigma_1\sigma_1, ..., \sigma_1^{n-1}, \sigma_3^n\}$, . . . ,
$\{\sigma_1, \sigma_1\sigma_1, ..., \sigma_1^{n-1}, \sigma_n^n\}, \{\sigma_1, \sigma_1\sigma_1, ..., \sigma_2^{n-1}, \sigma_1^n\}$, $\{\sigma_1, \sigma_1\sigma_1, ..., \sigma_2^{n-1}, \sigma_2^n\}$, . . . ,
$\{\sigma_1, \sigma_n\sigma_n, ..., \sigma_n^{n-1}, \sigma_n^n\}, \{\sigma_2, \sigma_1\sigma_1, ..., \sigma_1^{n-1}, \sigma_1^n\}$, $\{\sigma_2, \sigma_1\sigma_1, ..., \sigma_1^{n-1}, \sigma_2^n\}$, . . . ,
$\{\sigma_n, \sigma_n\sigma_n, ..., \sigma_n^{n-1}, \sigma_n^n\}$, $\{\sigma_1, \sigma_1\sigma_1, ..., \sigma_1^n\}$.

Since the number of configurations can be super-polynomial, a Hamiltonian cycle described simply by an ordered list of the configurations may be super-polynomial too. It is thus not clear at all whether Hamiltonicity in higraphs with intersections and no Cartesian products is in NP. We leave this as an open problem.

Similarly, the number of configurations in higraphs without intersections but with Cartesian products can be exponential in the number of blobs, and hence, for these higraphs too, it is not clear whether the Hamiltonicity problem is in NP. For example, if we connect by a single undirected edge each pair of atomic blobs within each of the orthogonal components in Fig. 7, we end up with a Hamiltonian higraph of $O(n)$ blobs, $O(n)$ edges and $O(2^n)$ configurations.

5.2 Pseudo Hamiltonian cycles of configurations

The simple 'all-to-all' semantics we use in most of this paper allows a configuration to have more than one successor by the same edge. This fact motivates the definition of a cycle in which more than one of the successor configurations of an edge can appear in the cycle. When we allow this, the 'cycle' no longer looks like a cycle.

Definition 14 *A* pseudo cycle of length n *in a higraph is an ordered list of n pairs, each consisting of a configuration and an edge that leaves it. The first and last elements of the list are identical, and the following properties hold:*

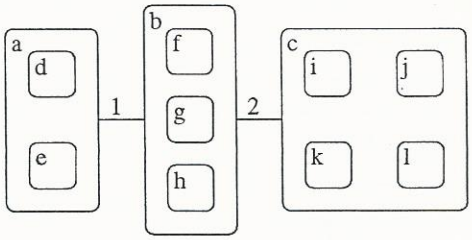

Figure 17: A pseudo Hamiltonian higraph that is not Hamiltonian

1. *If C is the configuration of the i'th pair on the list, for $1 < i \leq n$, there exists j, $1 \leq j < i$, such that if (C', e) is the j'th pair then C is a successor of C' by the edge e.*

2. *If (C, e) is the i'th pair on the list, for $1 \leq i < n$, there exists j, $i < j \leq n$, such that the configuration C' of the j'th pair is a successor of C by the edge e.*

A pseudo Hamiltonian cycle *for a higraph with m configurations is a pseudo cycle of length $m + 1$, in which every configuration, except the first (which is the same as the last), appears exactly once.*

In ordinary graphs the definitions of a Hamiltonian cycle and a pseudo Hamiltonian cycle coincide. Here each Hamiltonian cycle of configurations can be represented as a pseudo Hamiltonian cycle, but not necessarily vice versa. Fig. 17 is an example of a higraph with a pseudo Hamiltonian cycle of configurations, but containing no Hamiltonian cycle of configurations. The pseudo Hamiltonian cycle is: $(\{b, f\}, 2), (\{c, i\}, 2), (\{c, j\}, 2), (\{c, k\}, 2), (\{c, l\}, 2), (\{b, g\}, 1), (\{b, h\}, 1), (\{a, d\}, 1), (\{a, e\}, 1), (\{b, f\}, 2)$.

The definition of a pseudo Hamiltonian cycle can be extended to higraphs containing intersections or Cartesian products in an obvious way.

All we can say about the complexity of detecting pseudo Hamiltonicity is the trivial fact that it is NP-hard, since a Hamiltonian cycle in an ordinary graph is a special case.

5.3 Hamiltonian cycles of blobs

It is possible to define a cycle of blobs in a higraph to be a list of blobs in which the first and last are identical, and in which each adjacent pair (a, b) has the property that b is a successor of a in the higraph. The successor relationship between blobs can be defined in several ways, the definition depending on the semantics being used. Possibilities include the existence of an edge between the blobs, the existence of an edge between any of the ancestors of the blobs, etc. A cycle of blobs can be defined without any limitations on the blobs appearing in the cycle, or with the restriction that they have to be elementary. Also, one may restrict the number of times an edge can be used in a cycle.

A Hamiltonian cycle of (elementary) blobs is defined as a cycle containing all the (elementary) blobs, such that each blob appears exactly once, except for the first blob that appears also as the last one.

For higraphs with no intersections and no Cartesian products, there exists a Hamiltonian cycle of elementary blobs if and only if there exists a Hamiltonian cycle of configurations. The problem of deciding whether a higraph contains a Hamiltonian cycle of blobs is NP-complete, since on the one hand ordinary graphs are a special case, and on the other hand it is possible to check whether a given cycle is Hamiltonian in time linear in the size of the higraph. The

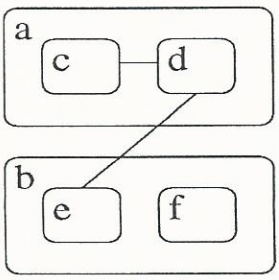

Figure 18: A bipartite higraph according to one definition but not according to the other

problem of deciding whether there is a Hamiltonian cycle of blobs such that the number of times an edge can be used is bounded, is also NP-complete.

6 Bipartition

A graph is bipartite if it is possible to partition its vertices into two subsets, such that every edge connects a vertex in one subset to a vertex in the other. There are two reasonable ways of extending this definition to higraphs. We shall provide them both, together with an algorithm to check whether a given higraph is bipartite. In this section, higraphs are assumed not to contain Cartesian products.

Definition 15 *1. A higraph H is* bipartite *if it is possible to partition H's blobs into two subsets B_1 and B_2, such that each edge of H connects a blob in B_1 to a blob in B_2.*

2. A higraph H is bipartite *if it is possible to partition H's blobs into two subsets B_1 and B_2, such that each edge of H connects a blob in B_1 to a blob in B_2, and for each blob b, if $b \in B_i$, for $i = 1, 2$, then all the descendants of b are in B_i.*

To see that there is a difference between the two definitions, consider Fig 18. According to the first definition the higraph is bipartite, since the blobs can be partitioned into two subsets with c and e in the first and d in the second. According to the second definition the higraph is not bipartite, because, on the one hand, blobs c and d have to be in the same subset since they have a common ancestor, but, on the other hand, there is an edge between them, so they have to be in different subsets. Thus, by the second definition, a higraph with an edge connecting two blobs that have a common ancestor or descendant cannot be bipartite.

The second definition is the one used (implicitly) in the project on security systems of [10, 11]. Although these authors do not talk about bipartition explicitly, the higraphs they use are bipartite; one part represents users, and the other represents files.

We now present algorithms to check whether a given higraph is bipartite. The same algorithm can be used for both definitions, the difference being only in the procedure 'SplitBlobs'.

$B_1, B_2, B_1', B_2' \leftarrow \emptyset$
Bipart $\leftarrow$ *true*
while $B_1 \cup B_2 \neq B$ and Bipart $=$ *true*
  if $B_1' \neq \emptyset$ **then**
    choose $x \in B_1'$
    $B_1' \leftarrow B_1' - \{x\}$

```
        Bipart ← SplitBlobs (x, B_1, B_2, B'_1, B'_2)
    else if B'_2 ≠ ∅ then
        choose x ∈ B'_2
        B'_2 ← B'_2 − {x}
        Bipart ← SplitBlobs (x, B_2, B_1, B'_2, B'_1)
    else
        choose x ∈ B − (B_1 ∪ B_2)
        B_1 ← B_1 ∪ {x}
        Bipart ← SplitBlobs (x, B_1, B_2, B'_1, B'_2)
    endif
  endif
end
if Bipart = true then   return ('The higraph is bipartite')
    else   return ('The higraph is not bipartite')
endif
```

Here is the procedure SplitBlobs for the first definition:

```
Procedure SplitBlobs (x, H_1, H_2, H'_1, H'_2) returns (code)
for each (x, y) ∈ E
    if y ∈ H_1 then   return (false)
    else if y ∉ H_2 then
        H_2 ← H_2 ∪ {y}
        H'_2 ← H'_2 ∪ {y}
    endif
  endif
end
return (true)
end procedure
```

In order to make the algorithm fit the second definition, add the following lines before **return** (*true*) at the end of SplitBlobs:

```
for each y such that anc(x, y)
    if y ∈ H_2 then
        return (false)
    else if y ∉ H_1 then
        H_1 ← H_1 ∪ {y}
        H'_1 ← H'_1 ∪ {y}
    endif
  endif
end
```

7 Minimum cover

There are many reasons to make graphical formalisms succinct, and higraphs are no exception. One issue that arises in higraphs is to minimize the number of edges, such that the successor relationship defined by the original higraph is preserved. We may be interested in preserving this relationship for blobs or for elementary blobs. We first deal with the case of elementary blobs.

Definition 16 *The problem of* minimum cover for (elementary) blobs *is defined as follows:*
Instance: A higraph $H = (B, E, \rho, \pi)$, and a positive integer k.
Question: Does E contain an edge-cover of size $\leq k$, i.e., a subset $E' \subseteq E$ with $|E'| \leq k$, such that every two (elementary) blobs connected by an edge $e \in E$ are connected also by an edge $e' \in E'$.

Proposition 3 *The minimum cover problem for elementary blobs is NP-complete.*

Proof: It is easy to see that the problem is in NP, since a nondeterministic algorithm need only guess a subset of E and then check in polynomial time the required property for any two elementary blobs.

For hardness, we reduce the classical minimum cover problem to minimum cover for elementary blobs. Recall that the classical minimum cover problem is defined as follows:
Instance: Collection C of subsets of a finite set S, and a positive integer $k \leq |C|$.
Question: Does C contain a cover for S of size $\leq k$, i.e., a subset $C' \subseteq C$ with $C' \leq k$, such that every element of S belongs to at least one member of C'?

Let $S = \{s_1, s_2, ..., s_n\}$, $C = \{c_1, c_2, ..., c_m\}$ and k, be an arbitrary instance of minimum cover. Without lost of generality, assume that $\bigcup_{1 \leq i \leq m} c_i = S$. Now define the corresponding higraph $H = (B, E, \rho)$ and the value k' for our problem of minimum cover for Elementary Blobs as follows:

$$B = \{x_1, x_2, ..., x_n\} \cup \{c'_1, c'_2, ..., c'_m\} \cup \{s\}$$

$$\forall 1 \leq i \leq n,\ 1 \leq j \leq m \quad (x_i \in \rho(c'_j) \text{ iff } s_i \in c_j)$$

$$E = \{(s, c'_j) : 1 \leq j \leq m\}$$

$$k' = k$$

There is a cover of size k' for the pairs of elementary blobs in H if and only if there is a cover of S of size k. Indeed, the pairs of elementary blobs created by the edges in E are exactly $(s, x_i), 1 \leq i \leq n$. Now, suppose that $E' = \{(s, c'_{j_l}) : 1 \leq l \leq k'\}$ is a cover for the pairs of elementary blobs. Then for each x_i there is $e = (s, c'_{j_l}) \in E'$ connecting s and x_i, and hence x_i is contained in c'_{j_l}, and the set c_{j_l} contains s_i. Thus, the set $C' = \{(c_{j_l} : 1 \leq l \leq k\}$ is a cover for S. Now, suppose that $C' = \{(c_{j_l} : 1 \leq l \leq k\}$ is a cover for S. For each $s_i, 1 \leq i \leq n$, there is a set c_{j_l} in C' containing s_i, and the blob x_i is contained in the blob c'_{j_l}. Thus the set $E' = \{(s, c'_{j_l}) : 1 \leq l \leq k'\}$ is a cover for the pairs of elementary blobs. ■

We now turn to the minimal cover problem for general blobs.

Proposition 4 *The minimum cover problem for blobs is NP-complete.*

Proof: That the problem is in NP is obvious, as in the previous proof.

Here too we carry out a reduction from the classical minimum cover problem to minimum cover for blobs. Let SZ, C and k be an instance of minimum cover as in the previous proof, and we make the same assumption on the union of the c_i. The higraph $H = (B, E, \rho)$ and the value k' for minimum cover for blobs are now defined by:

$$B = \{x_1, x_2, ..., x_n\} \cup \{c'_1, c'_2, ..., c'_m\} \cup \{u_1, u_2, ..., u_m\} \cup \{u'_1, u'_2, ..., u'_m\} \cup \{s\}$$

$$\forall j, 1 \leq j \leq m \;\; (u_j \in \rho(c'_j),\; u_j \in \rho(u'_j),\;\; \forall i, 1 \leq i \leq n \; (x_i \in \rho(c'_j) \iff s_i \in c_j)\;)$$

$$E = \{(s, c'_j), (s, u_j) : 1 \leq j \leq m\}$$

$$k' = m + k$$

There is a cover of size k' for the pairs of blobs in H if and only if there is a cover of S of size k. Indeed, the pairs of blobs created by the edges in E are exactly (s, x) for every blob $x \in B - \{s\}$. E' must contain the edge (s, u_j) for each j, $1 \leq j \leq m$, since otherwise there would exist a pair of type (s, u'_j) that is not covered by E'. Besides these edges, there is room for k additional edges in E'. These k edges, denote them by $\{(s, c'_{j_l}) : 1 \leq l \leq k'\}$, have to cover all the pairs of type (s, x_i). For each x_i, there is an $e = (s, c'_{j_l}) \in E'$ connecting s and x_i, and hence x_i is contained in c'_{j_l}, and the set c_{j_l} contains s_i. Thus, the set $C' = \{(c_{j_l} : 1 \leq l \leq k\}$ is a cover for S. Now, suppose that $C' = \{(c_{j_l} : 1 \leq l \leq k\}$ is a cover for S. For each s_i, $1 \leq i \leq n$, there is a set c_{j_l} in C' containing s_i, and the blob x_i is contained in the blob c'_{j_l}. Thus, the set $E' = \{(s, c'_{j_l}) : 1 \leq l \leq k\} \cup \{(s, u_j) : 1 \leq j \leq m\}$ is a cover for the pairs of blobs of size k'. ■

8 Discussion

Although higraphs are a generalization of graphs, there are properties that are less easily definable for higraphs. Planarity is one. We might say that a planar layout forbids edges to cross each other, but since blobs take up space, many such crossings can be replaced by edges that traverse blobs unnecessarily, and we would definitely not want to disallow edges that cross the contours of blobs, since these are a central feature of higraphs.

In higraphs we are more interested in the Venn-like question of whether the higraph can be drawn on the plane at all. Since blobs are not arbitrary simple closed curves, it may be impossible to lay out a set of blobs in such a way as to meet some particular set of containment requirements. Venn [12] has already shown that four circles c_1, c_2, c_3, c_4 cannot be drawn on the plane such that for any $A \subseteq \{1, 2, 3, 4\}$ and $\overline{A} = \{1, 2, 3, 4\} - A$, the intersection of $\bigcap_{i \in A} c_i$ with $\bigcap_{i \in \overline{A}} \overline{c_i}$ is non-empty. However, it is possible to draw four non-circular blobs, b_1, b_2, b_3, b_4, such that for any such A, the intersection of $\bigcap_{i \in A} b_i$ with $\bigcap_{i \in \overline{A}} \overline{b_i}$ is non-empty. Still, in [10] there are examples of simple containment relations that cannot be represented on the plane using blobs. One such example is the higraph with blobs a_i, b_i, $1 \leq i \leq 5$, such that for each $1 \leq i \leq 5$, $\rho(a_i) = \{b_j : j \neq i\}$.

Another term taken from the world of graphs that cannot be extended naturally to higraphs is a tree. One way of defining a conventional tree is that it is simply a connected acyclic graph [1]. Among the important properties of trees are the following:

1. A connected graph with n vertices is a tree if and only if it contains $n - 1$ edges.

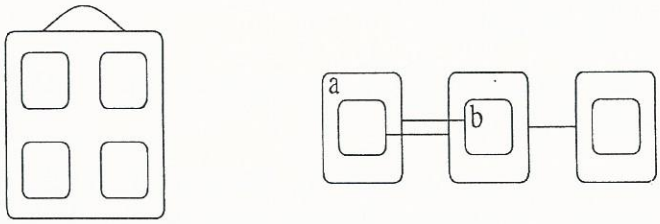

Figure 19: (a) a connected higraph containing no partial tree higraph; (b) a 'hitree' that does not disconnect when edge (a, b) is removed

2. A connected graph is a tree if and only if removing any edge disconnects it.

3. A graph G contains a partial graph which is a tree if and only if it is connected.

It seems reasonable for higraphs to define a *hitree* to be a connected acyclic higraph, where connectivity is between configurations and cycles are cycles of configurations. However, if we adopt this definition and in the above properties we replace each appearance of 'vertex' by 'configuration', the properties do not hold. Fig. 19(a) shows a connected higraph that contains no partial higraph which is a hitree. Thus, property 3 does not hold for higraphs. By adding edges to this higraph one can easily violate property 1 too. In Fig. 19(b) the higraph is a hitree, but removing edge (a, b) will not cause the higraph to become disconnected. Thus, property 2 does not hold either.

Thus, hitrees seem to be more subtle, and we leave open the issue of defining them properly.

Moreover, there are cases in which, given a set of blobs B and the functions ρ and π, it is impossible to find a set of edges E that forms a hitree. For example, consider the set of blobs $B = \{a, b, c, 1, 2, 3\}$ with the following hierarchy function: $\rho(a) = \rho(b) = \rho(c) = \{1, 2, 3\}$. If $E \neq \emptyset$ then the higraph contains a cycle of configurations, and hence it is impossible to find E such that (B, E, ρ, π) is a hitree. Higraphs with no intersections but with Cartesian products suffer from the same problem. For example, consider the set of blobs $B = \{1, a, b, c, d\}$, with $\rho(1) = \{a, b, c, d\}$ and $\pi_1(1) = \{a, b\}$ and $\pi_2(1) = \{c, d\}$. There is no set of edges E that forms a connected higraph with no cycles.

References

[1] M. Gondran, and M. Minoux, *Graphs and Algorithms*, John Wiley and Sons, 1984.

[2] E. Hammer , "Representing Relations Diagrammatically", *Working Papers on Diagrams and Logic* (G. Allwein and J. Barwise, eds.), Indiana University Logic Group, Preprint No. IULG-93-24, 1993, pp. 77–119.

[3] D. Harel, "Statecharts: A Visual Formalism for Complex Systems." *Sci. Comput. Prog.* 8 (1987), 231–274.

[4] D. Harel, "On Visual Formalisms", *Comm. Assoc. Comput. Mach.* **31** (1988), 514–530.

[5] D. Harel and E. Gery, "Executable Object Modeling with Statecharts", *Computer* (July 1997), 31–42.

[6] D. Harel, C-A. Kahana, "On Statecharts with Overlapping", *ACM Trans. Software Engineering and Methodology* **1** (1992), 399–421.

[7] D. Harel and A. Naamad, "The STATEMATE Semantics of Statecharts", *ACM Trans. Soft. Eng. Method.* **5**:4 (Oct. 1996), 293–333.

[8] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtul-Trauring and M. Trakhtenbrot, "STATEMATE: A Working Environment for the Development of Complex Reactive Systems", *IEEE Transactions on Software Engineering* **16** (1990), 403–414.

[9] D. Harel, A. Pnueli, J.P. Schmidt, and R. Sherman, "On the Formal Semantics of Statecharts", in *Proc. 2nd IEEE Symp. on Logic in Computer Science*, IEEE Press, New York, 1987, pp.54–64.

[10] A. Heydon, *Processing Visual Specifications of File System Security*, Doctoral Thesis, Carnegie-Mellon University, Pittsburgh, PA, 1992.

[11] M.W. Maimone, J.D. Tygar, and J.M. Wing, "Formal Semantics for Visual Specification of Security", in *Visual Languages and Visual Programming*, Plenum Press, New York, 1990, pp. 97–116.

[12] J. Venn, *Symbolic Logic*, 2nd ed., London, 1894. (Reprinted by Chelsea, Bronx, N.Y., 1971.)