Polychotomic Encoding: A Better Quasi-Optimal Bit-Vector Encoding of Tree Hierarchies

Robert E. Filman

RIACS Technical Report 02.02

April 2002

16th European Conference on Object-Oriented Programming (ECOOP-2002), Málaga, Spain, June, 2002

Polychotomic Encoding: A Better Quasi-Optimal Bit-Vector Encoding of Tree Hierarchies

Robert E. Filman, RIACS

RIACS Technical Report 02.02

April 2002

Polychotomic Encoding is an algorithm for producing bit vector encodings of trees. Polychotomic Encoding is an extension of the Dichotomic Encoding algorithm of Raynaud and Thierry. Polychotomic and Dichotomic Encodings are both examples of hierarchical encoding algorithms, where each node in the tree is given a *gene*—a subset of {1,…,*n*}. The encoding of each node is then the union of that node's gene with the genes of its ancestors. Reachability in the tree can then be determined by subset testing on the encodings.

Dichotomic Encoding restructures the given tree into a binary tree, and then assigns two bit, incompatible (chotomic) "genes" to each of the two children of a node. Polychotomic Encoding substitutes a multibit encoding for the children of a node when the restructuring operation of Dichotomic Encoding would produce a new heaviest child (child requiring the most bits to represent a tree of its children) for that node. The paper includes a proof that Polychotomic Encoding never produces an encoding using more bits than Dichotomic Encoding. Experimentally, Polychotomic Encoding produces a space savings of up to 15% on examples of naturally occurring hierarchies, and 25% on trees in the randomly generated test set.

This work was supported in part by the National Aeronautics and Space Administration under Cooperative Agreement NCC 2-1006 with the Universities Space Research Association (USRA).

This report is available online at http://www.riacs.edu/trs/

Polychotomic Encoding: A Better Quasi-Optimal Bit-Vector Encoding of Tree Hierarchies

Robert E. Filman

Research Institute for Advanced Computer Science
NASA Ames Research Center MS/269-2
Moffett Field, CA 94025 U.S.A.
rfilman@mail.arc.nasa.gov

Abstract. Polychotomic Encoding is an algorithm for producing bit vector encodings of trees. Polychotomic Encoding is an extension of the Dichotomic Encoding algorithm of Raynaud and Thierry. Polychotomic and Dichotomic Encodings are both examples of hierarchical encoding algorithms, where each node in the tree is given a *gene*—a subset of $\{1, \ldots, n\}$. The encoding of each node is then the union of that node's gene with the genes of its ancestors. Reachability in the tree can then be determined by subset testing on the encodings.
Dichotomic Encoding restructures the given tree into a binary tree, and then assigns two bit, incompatible (chotomic) "genes" to each of the two children of a node. Polychotomic Encoding substitutes a multibit encoding for the children of a node when the restructuring operation of Dichotomic Encoding would produce a new heaviest child (child requiring the most bits to represent a tree of its children) for that node. The paper includes a proof that Polychotomic Encoding never produces an encoding using more bits than Dichotomic Encoding. Experimentally, Polychotomic Encoding produces a space savings of up to 15% on examples of naturally occurring hierarchies, and 25% on trees in the randomly generated test set.

1 Introduction

Bit-vectors encodings are a popular mechanism for quickly determining reachability in a directed graph. Let the reachability relation be $\prec\!\!\!\prec$. A bit-vector encoding is a function, γ, from the nodes of the graph to a subset of $\{1, \ldots, n\}$. With a bit-vector encoding, $x \prec\!\!\!\prec y \iff \gamma(y) \subseteq \gamma(x)$. Figure 1 shows a bit-vector encoding of a simple tree. Since subsets of integers can be represented by bit vectors and bit-vector subset testing requires only a few instructions on most hardware, bit vector encodings can be both a time and space efficient way of testing reachability. Reachability is important for object-oriented programming languages, as determining if an element of direct class X can be safely cast to class Y is frequently required. Similarly, method dispatch in object-oriented languages requires dynamic subclass testing [2, 7, 8, 10]. Membership testing is also found in many AI applications [6, 13]. Directed graph reachability is a common problem throughout Computer Science.

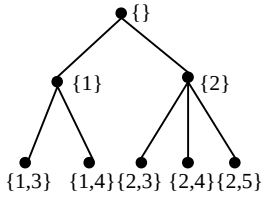

Fig. 1. A bit-vector encoding

Bit-vector encodings have been a subject of much research. Good algorithms quickly determine an encoding for a graph using as few as bits as possible. Bit-vector encoding algorithms divide into algorithms over trees versus all acyclic graphs, and algorithms that demand the entire graph before deciding the encoding (static algorithms) versus algorithms that can incrementally modify the encoding on the addition of new nodes and relations (dynamic algorithms).

Here we are concerned with algorithms encoding static trees. This paper describes the *Polychotomic Encoding* (PE) algorithm. Polychotomic Encoding combines elements of both the Dichotomic Encoding (DE) encoding of Raynaud and Thierry [11] and the 2-dimensional (multibit) encoding of Caseau et al. [4] (CHNR). Succinctly, Polychotomic Encoding performs Dichotomic Encoding of a node's children until doing so would create a new heaviest child (child requiring the most bits to represent a tree of its children), and then uses a multibit encoding to encode the remaining children. Polychotomic Encoding has the same time and space complexity as Dichotomic Encoding. Section 8 contains a proof that Polychotomic Encoding never produces an encoding requiring more bits than Dichotomic Encoding. Experimentally, Polychotomic Encoding produces a savings of up to 15% over Dichotomic Encoding on naturally occurring examples of hierarchies, and up to 25% on some randomly generated trees.

2 Hierarchies

A hierarchy, H is a set T and a transitive, reflexive and anti-symmetric relation $\preccurlyeq$, $H = (T, \preccurlyeq)$. The *transitive reduction* of H, $x \prec y$, is defined as $x \preccurlyeq y, x \neq y$, and $\neg\exists z.x \preccurlyeq z \preccurlyeq y, x \neq z \neq y$. Often, given T and the various $\prec$ relationships, one infers $\preccurlyeq$ by closing over $\prec$ (extended with $x \preccurlyeq x$). A common example of a hierarchy is subclass relationships in object-oriented languages, where $\prec$ is the parent-child relation.

The computational problem is to build a representation and algorithm that can answer the question $x \overset{?}{\preccurlyeq} y$. The developer of such an algorithm can perform tradeoffs about the time taken to build the representation, the space required by the representation, the time needed to perform the test, and the cost of modifying the representation dynamically if new elements of T or new $\prec$ relationships are

asserted. Some algorithms apply to arbitrary partial orders (multiple inheritance, or MI), while others are restricted to trees (single inheritance, or SI).

Hierarchies are important in object-oriented systems technology, because object types over subclass form a hierarchy, and the $\stackrel{?}{\prec\!\!\prec}$ test is used to determine if one object can be viewed as an instance of another class. For this reason, this is sometimes called the *subtyping problem*, though the results are clearly applicable to any partial order. Subtyping is a frequent operation in the compiled code of object systems. Zibin and Gil provide a good discussion of the space of tradeoffs and the impact of the performance of the subclassing test in OO systems [17].

3 Encodings

In a bit-vector encoding, the root of the tree is assigned the null set. Every other node in the hierarchy has some non-null subset of $\{1, \ldots, n\}$ (its *gene*) associated with that node. The encoding of a node (γ) is the union of the gene of the node with the encoding of its parents. Thus, the encoding of a node is the union of the gene of a node with genes of its ancestors. Such a coding scheme is called a hierarchical encoding.

Figure 2, adapted from [11], shows three different representations of a bit-vector encoding. The *complete code* lists the elements of the set $\{1, \ldots, n\}$ used to encode each node. Since the code is hierarchical, the encoding can also be represented in the *reduced code* as just the gene of the node. The complete code can be obtained by taking the union this gene with the genes of its ancestors. The *bit-vector* encoding is obtained by associating a one in a vector of bits with the corresponding elements of the complete code. The three are equivalent representations: each can be straightforwardly derived from any other. This close correspondence between the bit-vector encoding and the numeric encodings leads us to refer to individual numbers in an encoding as *bits*.

A set of genes are *chotomic* if no element of the set is a subset of another. The genes $\{1\}$ and $\{2\}$ (represented as the two bit bit-vectors [10] and [01]) are a two-element, two bit chotomic set; the genes $\{1, 2\}$, $\{1, 3\}$, $\{1, 4\}$, $\{2, 3\}$, $\{2, 4\}$, and $\{3, 4\}$ ([1100], [1010], [1001], [0110], [0101], and [0011]) represent a set of six four-bit chotomic genes.

In general, our desired relationship

$$x \prec\!\!\prec y \iff \gamma(y) \subseteq \gamma(x)$$

is always true of a hierarchical encoding if the sibling nodes in the tree have chotomic genes, and the bits of these genes are not reused in the genes of any of the descendants of the siblings. (The bits can, however, be safely reused in the cousins of the siblings.) The forward part of this equivalence:

$$x \prec\!\!\prec y \Rightarrow \gamma(y) \subseteq \gamma(x)$$

is true by the definition of a hierarchical encoding—since $\gamma(x)$ includes all the bits of its ancestors, γ of any ancestor, y, is a subset of $\gamma(x)$. The reverse part

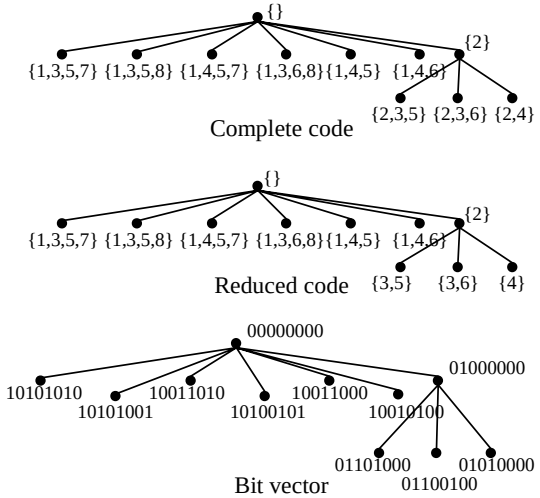

Fig. 2. Three representations of bit-vector encoding

of the equivalence

$$\gamma(y) \subseteq \gamma(x) \Rightarrow x \not\prec y$$

can be seen to be true by a proof by contradiction. Assume that in our tree, $\gamma(y) \subseteq \gamma(x)$ and y is not an ancestor of x. (x cannot be a proper ancestor of y, as each level of the tree extends γ by a non-null gene.) Since this is a tree, y and x have a least common ancestor, z, and each of y and x have ancestors (or y or x itself) y' and x' that are siblings and children of z. The genes of $\gamma(y')$ and $\gamma(x')$ are chotomic—neither is a subset of the other, and the gene of each is not included in its sibling's descendants. Hence, there is at least one bit, b, of the gene of y' that is not in the gene of x'. Since the genes of chotomic siblings can't be the genes of their descendants, this number, b, is not in $\gamma(x)$. But this contradicts our assumption that $\gamma(y) \subseteq \gamma(x)$.

4 Prior Work

The naive approach to bit vector encoding associates a unique single bit gene with each type (node). This then requires $|T|^2$ space—a vector of $|T|$ bits for each of the $|T|$ nodes. While in the worst case (a tree of single-child nodes), this space complexity is unavoidable, several researchers have developed algorithms that are usually considerably more space efficient.

Aït-Kaci et al. [1] proposed a modulation algorithm, which started by assigning a unique single-bit gene to each node of the tree and then applied a repeated binary splitting and dichotomic coding to the recursive subtrees.

Another early study of bit vector encodings for MI hierarchies was Caseau [3]. That work used single bit genes. Caseau's algorithm was based on embedding the partial order in a lattice, determining which nodes have incompatible encodings, and parceling out gene assignments as a search process, where failure can prompt a change in earlier decisions. Unfortunately, completing the lattice from a partial order can take exponential time.

Vitek, Horspool and Krall [14] extended this approach. Like Caseau, they constructed a conflict graph and colored this graph, using one bit for each gene. However, to get a better encoding, they preceded this step by "balancing" the original hierarchy. One of the results that came out of this work was the recognition that that the gene allocation problem can be expressed as a graph coloring problem. Thus, the literature of graph-coloring algorithms is applicable to the bit-vector encoding problem.

Caseau et al presented a method [4] centered on conflict graphs, but also included an algorithm for trees. The key idea of that work was to use multi-bit genes for distinguishing the children of a node.

Most relevant to the current paper is the work of Raynaud and Thierry on Dichotomic Encoding [11]. Dichotomic Encoding performs a quick balancing that experimentally often produces shorter encodings. Dichotomic Encoding is discussed in detail in Section 5.

There are also many non–bit-vector approaches to the subtyping problem. The most straightforward way of computing $\preccurlyeq^2$ is to literally represent the relation in a binary matrix. This requires space proportional to $|T|^2$ and provides access in constant time. At the opposite extreme is the graph encoding, where only the $\prec$ relations are stored and the system dynamically searches to establish the path at each request. This requires space only proportional to the number of links (in an SI system, $|T|$), but work at runtime(in SI systems) proportional to the height of the graph for each $\preccurlyeq^2$ test.

Algorithms for efficiently minimizing the space needed to represent inheritance hierarchies have been a fertile area for research. For SI hierarchies, *relative numbering* (Schubert numbering) [12] associates with each node of the tree, a, its index, i_a in a preorder traversal. Relative numbering also stores its upper bound, u_a, the maximum index of its descendants (nodes for which $x \preccurlyeq a$ is true). With Schubert numbering,

$$x \preccurlyeq a \iff i_a \leq i_x \leq u_a.$$

Cohen's algorithm [5], for a tree of height h, stored for each node, a, both its depth in the hierarchy (distance from the root), d_a, and an array, p_a of h elements. It stored each of its ancestors in this array, putting an ancestor x at the d_x element of this array. For this algorithm,

$$a \preccurlyeq x \iff p_a[d_x] = x.$$

Vitek, Horspool and Krall [14] generalized Cohen's algorithm to Bit-Packed Encoding (BPE) which handled multiple inheritance by slicing, partitioning T into chunks. Recently, Zibin and Gil [17] extended the ideas of BPE with a more efficient slicing algorithm based on PQ-trees.

5 Dichotomic Encoding

At the last ECOOP, Raynaud and Thierry [11] presented *Dichotomic Encoding*, a quick (linear in $|T|$ times $k \log k$ in the branching factor of the tree) algorithm for creating a bit vector encoding. Dichotomic Encoding transforms the initial hierarchy into a binary tree by introducing new nodes, and giving 2-bit, chotomic (that is, dichotomic) genes to the children in the transformed tree. The transformation is driven by the goal of balancing the binary tree, so that the number of bits required to represent the two children of a node are both relatively low and equal.

In general, chotomic encoding algorithms are given a node, x, and a value, $nextFreeBit$ of the next free integer for assignment. The algorithm determines the number of bits needed to code the number of children of x (`cHat`), which we call $neededBits$. It then assigns chotomic genes to its children using the values $nextFreeBit, nextFreeBit + 1, \ldots, nextFreeBit + neededBits - 1$, and then recursively encodes its children starting at $nextFreeBit + neededBits$. Figure 3 illustrates this is pseudo-code, where `code(n, k)` returns chotomic sets of k elements based at n.

```
chotomic(node x, int nextFreeBit)
{
    int neededBits = cHat (x.children.size);
    for child in x.children
     as c in code (nextFreeBit, neededBits)
    {
        child.gene = c;
        chotomic (child, nextFreeBit + neededBits);
    }
}
```

Fig. 3. Chotomic encoding algorithm.

CHNR is a chotomic algorithm that works on the original tree and uses as many bits as needed to distinguish the children of a node. Dichotomic Encoding relies on restructuring the tree into a binary tree before applying the chotomic algorithm, with the goal of reducing the number of bits needed. For a node x with children $[a, b, \ldots]$, the algorithm first (recursively) computes the weight (number of bits needed to represent) of each child. Nodes with no children have weight 0. Nodes with one child, weigh 1 more than that child. Nodes with two children weigh 2 more than the heavier child. For a node with more than two children, the algorithm sorts the children by weight and selects the two "lightest" children, call them a (the lightest) and b (the second lightest). It then constructs a new node y (of weight $b + 2$), changes the parentage of a and b to be y, and inserts y into the child-set of x in their place. The algorithm iterates this process until x has only two children. The algorithm then uses the chotomic encoding

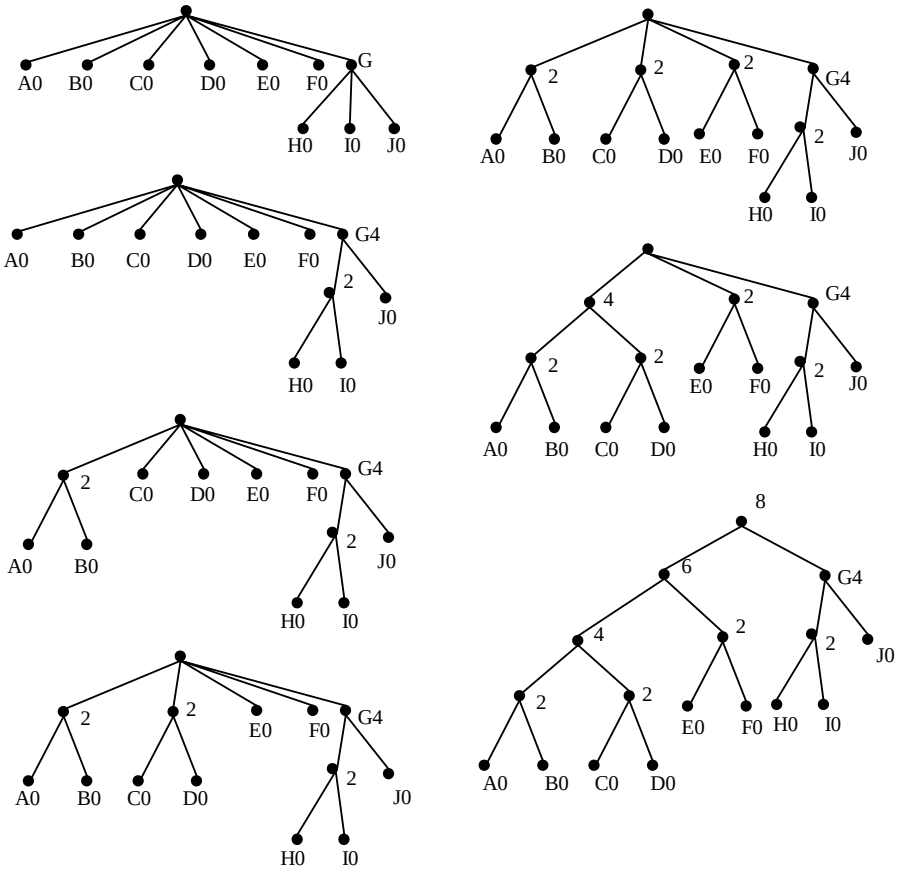

Fig. 4. Dichotomic Encoding with weights

algorithm (Figure 3) to assign the actual genes. Thus, the tree is a binary tree; except for the root, every node has one bit of two in its gene; and the genes of siblings are chotomic, hence, Dichotomic Encoding. The encoding of a node is then the union of its gene with the genes of its ancestors.

Figure 4 illustrates the Dichotomic Encoding process. That figure shows the emerging binary tree with the computed weights of the nodes. In their ECOOP paper, Raynaud and Thierry proved that this algorithm produces the minimal size dichotomic encoding. Dichotomic encoding has the virtues of being fast, easy to code, and of producing reasonably compact encodings. Raynaud and Thierry report that on benchmark OO hierarchies, Dichotomic Encoding produced a 13–36% improvement over the previously best algorithm, VHK.

In the discussion that follows, the elements of a bag S are $[x_1, \ldots, x_n]$, where the x_i are sorted. Thus, x_1 of a bag is the smallest element of that bag; the x_2 is the second smallest. The rightmost element of a bag is the largest. The use of two ellipses (e.g., $[x_1, \ldots x_i \ldots x_n]$) indicates that we can't specify where an element goes in the bag's sort. The cardinality of S is $|S|$. All logarithms are in base 2.

More formally, the behavior of Dichotomic Encoding is illuminated by defining $\mathcal{D}(S)$, the function that takes a bag of child weights and computes the weight of a node. $\mathcal{D}(S)$ is computed as

$$\mathcal{D}(S) = \begin{cases} 0 & |S| = 0 \\ 1 + x_1 & |S| = 1 \\ 2 + x_2 & |S| = 2 \\ \mathcal{D}([x_3, \ldots, x_2 + 2, \ldots, x_n]) & |S| > 2 \end{cases}$$

6 Polychotomic Encoding

For some graphs, Dichotomic Encoding is strikingly inefficient. For example, a parent with, say, ten equal weight children requires 8 bits in Dichotomic Encoding to differentiate its children. The multiple-bit encoding of CHNR needs only 5. Dichotomic is particularly clever when a node has a child that weighs a lot more than its other children—it efficiently combines these other children into a subtree. When the weight of that subtree is dwarfed by the weight of the heaviest child, the weight of the parent is just two more than that heaviest child's weight. On the other hand, Dichotomic Encoding gets into trouble with nodes that have more than a few equally heavy child nodes—it tends to find itself in an escalation of biggest child weights as it combines its children.

Polychotomic Encoding ameliorates this problem. It behaves like Dichotomic Encoding for nodes with zero, one or two children. Before performing the joining step of Dichotomic, where the two smallest children are combined into a single node of weight two more than the heavier, it checks to make sure that doing so would not create a new heaviest child. If it doesn't, like Dichotomic Encoding, it builds the new node and iterates. If it would, it stops joining children and uses a multi-bit, chotomic (CHNR) encoding for all the remaining children. Figure 5 illustrates the behavior of the Polychotomic algorithm.

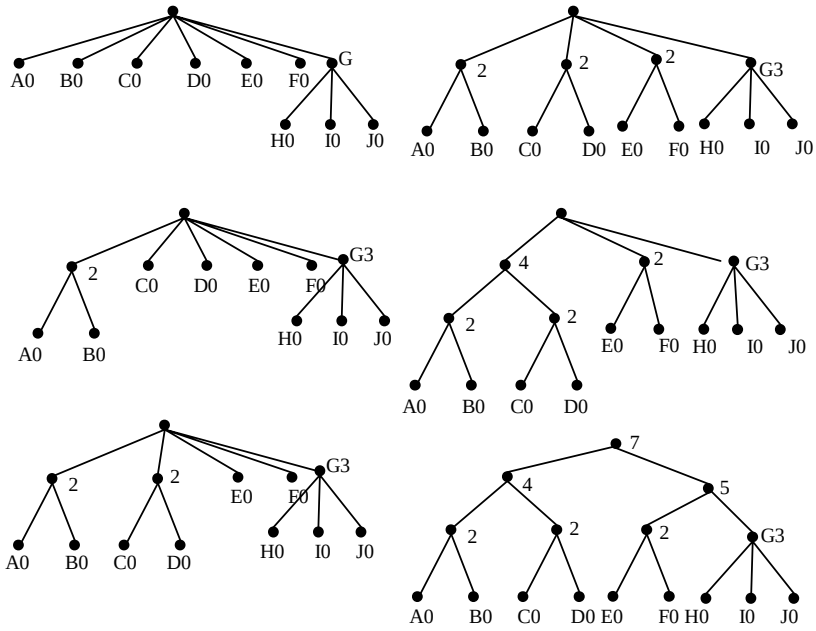

Fig. 5. Polychotomic Encoding

The function $\mathcal{C}$ defines the number of different chotomic genes using n bits:

$$\mathcal{C}(n) = \binom{n}{\lfloor n/2 \rfloor} = 2^{\lfloor n/2 \rfloor} \prod_{1 \le i \le \lceil n/2 \rceil} (2i-1)/i$$

Note that adding two bits to a multibit encoding covers three to four times as many cases:

$$\forall n. 3\mathcal{C}(n) \le \mathcal{C}(n+2) \le 4\mathcal{C}(n).$$

The function $\hat{c}(n)$ is the "inverse" of $\mathcal{C}$—the number of bits needed to create n different chotomic genes.

$$\hat{c}(n) = \text{the smallest } k \text{ such that } \mathcal{C}(k) \ge n.$$

Thus, $\hat{c}(9) = \hat{c}(10) = 5$; $\hat{c}(11) = 6$.

Once again, the structure of the algorithm is revealed in its weight function, $\mathcal{P}(S)$. The Polychotomic Encoding weight function is

$$\mathcal{P}(S) = \begin{cases} 0 & |S| = 0 \\ 1 + x_1 & |S| = 1 \\ 2 + x_2 & |S| = 2 \\ \mathcal{P}([x_3, \ldots, x_2 + 2, \ldots, x_n]) & |S| > 2, x_2 + 2 \le x_n \\ x_n + \hat{c}(n) & |S| > 2, x_2 + 2 > x_n \end{cases}$$

7 Computational Complexity

The computational complexity of Polychotomic Encoding is the same as that of Dichotomic Encoding. Each algorithm requires work proportional to $|T|$; the sorting step implies that for a tree with a largest branching factor of k, $k \log k$ additional steps may be needed at each node. (The actual assignment of bits to each gene can be done in time proportional to k; memoization can be used to pay this charge only once for each value of k.) That the worst case behavior of Polychotomic Encoding is the same as Dichotomic Encoding is not surprising, as Polychotomic Encoding algorithm is just a pruning of some of the work done under Dichotomic Encoding. The pruning is itself inexpensive (just a comparison at each split step), and for some trees, no pruning takes place at all.

In the worst case, for a "straight line" hierarchy composed single-child nodes, bit vector encoding needs one bit for each node except the root. Thus, the worst case space complexity of all bit vector encoding algorithms is proportional to $|T|^2$. In practice, the required space seems more on the scale of $|T| \log(|T|)$.

8 Polychotomic Encoding Is Never Worse than Dichotomic Encoding

This Section is a proof that Polychotomic Encoding never produces an encoding using more bits than Dichotomic Encoding. (Of course, it often produces one that uses fewer.) This is equivalent to showing that for bags of child weights, S,

$$\forall S. \mathcal{P}(S) \leq \mathcal{D}(S).$$

Definition 1. *A* flat *bag is a bag in which* $x_2 = x_n$ *or* $x_2 = x_n - 1$.

That is, in a flat bag, the second smallest element is either equal to the largest, or one less than the largest. It follows that all the elements of a flat bag except the smallest are either x_n or $x_n - 1$. Flat bags are the key place where PE and DE differ: for non-flat bags, each builds a subnode and recurses.

Lemma 1. *For a flat bag of cardinality a power of 2,* $S = [x_1, \ldots, x_n], n = 2^z$

$$\mathcal{D}(S) = x_n + 2z = x_n + 2 \log n.$$

Proof by induction over the value of z. For $z = 1, S = [x_1, x_2]$, by the definition of $\mathcal{D}, \mathcal{D}(S) = x_2 + 2$.

Assume the lemma is true of $(z - 1)$.

$$\begin{aligned}
&\mathcal{D}([x_1, x_2, \ldots, x_{n-1}, x_n]) \\
&\quad = \mathcal{D}([x_3, x_4, \ldots, x_n, x_2 + 2]) \\
&\quad = \mathcal{D}([x_5, x_6, \ldots, x_n, x_2 + 2, x_4 + 2]) \\
&\qquad\qquad \vdots && \text{After } n/2 \text{ steps} \\
&\quad = \mathcal{D}([x_2 + 2, x_4 + 2, \ldots, x_{n-2} + 2, x_n + 2]) && \text{A bag of size } 2^{z-1} \\
&\quad = (x_n + 2) + 2 \log(n/2) && \text{Induction assumption} \\
&\quad = x_n + 2 \log n
\end{aligned}$$

Lemma 2. *For a flat bag* $S = [x_1, \ldots, x_n], n \geq 3$,

$$\mathcal{D}(S) = x_{2(n-2^{\lfloor \log(n-1) \rfloor})} + 2\lceil \log n \rceil$$

Proof, by strong induction, over the size of the bag S.

For a flat bag of size 3,

$$\begin{aligned}
\mathcal{D}([x_1, x_2, x_3]) &= \mathcal{D}([x_3, x_2 + 2]) \\
&= x_2 + 4 \\
&= x_2 + 2\lceil \log 3 \rceil \\
&= x_{2(3-2^{\lfloor \log(3-1) \rfloor})} + 2\lceil \log 3 \rceil
\end{aligned}$$

Assume the theorem is true for all flat bags up to size $n-1$. We have three cases: (1) even n that is a power of 2, (2) even n not a power of two, and (3) odd n.

Case 1. Even n that is a power of 2

$$\begin{aligned}
&\mathcal{D}([x_1, x_2, \ldots, x_{n-1}, x_n]) \\
&\quad = x_n + 2\log n && \text{Lemma 1} \\
&\quad = x_n + 2\lceil \log n \rceil && \text{For powers of 2, } \lceil \log n \rceil = \log n \\
&\quad = x_{2(n-n/2)} + 2\lceil \log n \rceil \\
&\quad = x_{2(n-2^{\lfloor \log(n-1) \rfloor})} + 2\lceil \log n \rceil
\end{aligned}$$

For cases 2 and 3, after $\lfloor n/2 \rfloor$ recursions, the argument bag is once again flat, allowing application of the induction assumption.

Case 2. Even n, not a power of 2

$$\begin{aligned}
&\mathcal{D}([x_1, x_2, \ldots, x_{n-1}, x_n]) \\
&\quad = \mathcal{D}([x_3, x_4, \ldots, x_n, x_2 + 2]) \\
&\quad = \mathcal{D}([x_5, x_6, \ldots, x_n, x_2 + 2, x_4 + 2]) \\
&\qquad\qquad \vdots && \text{After } n/2 \text{ steps,} \\
&\quad = \mathcal{D}([x_2 + 2, x_4 + 2, \ldots, x_{n-2} + 2, x_n + 2]) && \text{A flat bag of } n/2 \text{ elts.} \\
&\quad = x_{2 \cdot 2(n/2-2^{\lfloor \log(n/2-1) \rfloor})} + 2 + 2\lceil \log(n/2) \rceil && \text{Induction, as the } k^{\text{th}} \\
&&& \text{elt. of this set is } x_{2k} \\
&\quad = x_{2(n-2^1 \cdot 2^{\lfloor \log(n/2-1) \rfloor})} + 2 + 2\lceil \log(n/2) \rceil \\
&\quad = x_{2(n-2^1 \cdot 2^{\lfloor \log(n-1) \rfloor - 1})} + 2 + 2\lceil \log(n/2) \rceil \\
&\quad = x_{2(n-2^{\lfloor \log(n-1) \rfloor})} + 2\lceil \log(n) \rceil
\end{aligned}$$

Case 3. Odd n

$$\begin{aligned}
&\mathcal{D}([x_1, x_2, \ldots, x_{n-1}, x_n]) \\
&\quad = \mathcal{D}([x_3, x_4, \ldots, x_n, x_2 + 2]) \\
&\quad = \mathcal{D}([x_5, x_6, \ldots, x_n, x_2 + 2, x_4 + 2])
\end{aligned}$$

$$
\begin{aligned}
&\vdots && \text{After } \lfloor n/2 \rfloor \text{ steps} \\
&= \mathcal{D}([x_n, x_2+2, \ldots, x_{n-3}+2, x_{n-1}+2]) && \text{A flat bag of} \\
& && n/2+1 \text{ elements.} \\
&= x_{2\cdot 2((n+1)/2-2^{\lfloor \log((n+1)/2-1)\rfloor})-2} + 2 + 2\lceil \log((n+1)/2)\rceil \\
& && \text{Induction, as for } k \geq 2, \\
& && \text{the } k^{\text{th}} \text{ elt. of this} \\
& && \text{set is } x_{2k-2} \\
&= x_{2((n+1)-2\cdot 2^{\lfloor \log((n+1)/2-1)\rfloor}-1)} + 2 + 2\lceil \log((n+1)/2)\rceil \\
&= x_{2(n-2\cdot 2^{\lfloor \log((n+1)/2-1)\rfloor})} + 2\lceil \log n\rceil \\
&= x_{2(n-2^{\lfloor \log(n-1)\rfloor})} + 2\lceil \log n\rceil
\end{aligned}
$$

Lemma 3.

$$x_n - 1 + 2\lceil \log n\rceil \leq \mathcal{D}([x_1, \ldots, x_n]) \leq x_n + 2\lceil \log n\rceil$$

This follows from Lemma 2, as $n - 2^{\lfloor \log(n-1)\rfloor} \geq 1$, and for flat bags, $\forall i. i \geq 2 \Rightarrow x_n - 1 \leq x_i \leq x_n$.

Theorem 1. $\forall S. \mathcal{P}(S) \leq \mathcal{D}(S)$

Proof by induction.

For $|S| \leq 2, \mathcal{P}(S) = \mathcal{D}(S)$.

For $|S| = 3$, we have the bag $S = [x_1, x_2, x_3]$. If $x_2 + 2 \leq x_3, \mathcal{P}(S) = \mathcal{D}(S)$. If $x_2 + 2 > x_3$,

$$\mathcal{D}(S) = \mathcal{D}([x_3, x_2+2]) = x_2 + 4$$

$$\mathcal{P}(S) = x_3 + \hat{c}(3) = x_3 + 3$$

If $x_2 + 2 > x_3$, then $x_2 + 4 > x_3 + 2$, and (since we're dealing with integers here), $x_2 + 4 \geq x_3 + 3$. So for bags of cardinality 3, $\mathcal{P}(S) \leq \mathcal{D}(S)$. Table 1 shows the results of $\mathcal{D}$ and $\mathcal{P}$ for flat bags of size up to 17. Keep in mind that for $i \geq 2, x_i$ is at most one less than x_n.

Bag Size	**DE**	**PE**	**Bag Size**	**DE**	**PE**
2	x_2+2	x_2+2	10	x_4+ 8	x_{10}+5
3	x_2+4	x_3+3	11	x_6+ 8	x_{11}+6
4	x_4+4	x_4+4	12	x_8+ 8	x_{12}+6
5	x_2+6	x_5+4	13	x_{10}+ 8	x_{13}+6
6	x_4+6	x_6+4	14	x_{12}+ 8	x_{14}+6
7	x_6+6	x_7+5	15	x_{14}+ 8	x_{15}+6
8	x_8+6	x_8+5	16	x_{16}+ 8	x_{16}+6
9	x_2+8	x_9+5	17	x_2+10	x_{17}+6

Table 1. Dichotomic and Polychotomic Encoding for small flat bags

Assume the theorem is true for bags of size up to $n-1$. There are two possibilities, either S is not flat, or flat. If S is not flat, $\mathcal{D}(S) = \mathcal{D}([x_3, \ldots, x_2 + 2, \ldots, x_n])$ and $\mathcal{P}(S) = ([x_3, \ldots, x_2 + 2, \ldots, x_n])$. We've reduced the problem to comparing $\mathcal{P}$ and $\mathcal{D}$ on the same bag of size $n-1$, so by the induction assumption, the theorem follows.

If S is flat, from Lemma 3, $\mathcal{D}(S) \geq x_n - 1 + 2\lceil \log n \rceil$ and $\mathcal{P}(S) = x_n + \hat{c}(n)$. Table 1 shows that by bags of size 16, $\mathcal{P}$ is already two bits smaller than $\mathcal{D}$. For larger bags, where $\mathcal{D}$ is more than a couple of bits larger than $\hat{c}$, with each additional 2 bits, $\mathcal{D}$ allows us to cover twice as many elements. However, with an additional 2 bits, $\hat{c}$ allows us to cover $2(2i-1)/i$ (or between three and four times as many) elements. On flat bags, $\mathcal{D}$ is thus larger. In the limit, $\mathcal{D}(S) = 2\mathcal{P}(S)$; for large flat bags, $\mathcal{D}$ uses almost twice as many bits.

9 Greedy Polychotomic Encoding

Polychotomic Encoding is based on the idea of doing Dichotomic Encoding until it would create a new heaviest child, and then switching to a multi-bit encoding. But consider the tree of Figure 2. Dichotomic Encoding requires 8 bits; Polychotomic Encoding 7 bits, and CHNR (pure multibit encoding) also 7. However, multibit encoding the six single leaves and then dichotomically combining the resulting node with the multibit encoding of the rightmost node yields a 6 bit code (Figure 6). This suggests the Greedy Polychotomic Encoding (GPE) algorithm: combine as many small children as possible using a multi-bit encoding until doing so would create a new heaviest child, and then use a multibit encoding on the remainder. Experiments with variations on this theme—splitting some of the children of a node into a subtree for multibit encoding, then recursively processing the child set—failed to find any algorithm that was consistently better than PE. While we can not recommend greedy PE algorithms, we include GPE in the results of Section 10 for the sake of comparison.

10 Experimental Results

There are three ways to experimentally evaluate algorithms: by examining their results on "natural" hierarchies, consistent structures (for example, complete trees of a given branching factor and depth), and by their performance on random trees. Natural hierarchies occur in the class structures of object-oriented programs, in the hierarchies of AI systems that attempt to model the world, in databases, and in other computer applications. This experiment compared the CHNR, Dichotomic Encoding, Polychotomic Encoding and Greedy Polychotomic Encoding algorithms. For programming class hierarchies, five examples are shown: VisualWorks2 and Digitalk3, SMALLTALK-80 class libraries; types extracted from the NextStep libraries; the ET++ graphical user interface; and the Java 1.3 class library. The first four of these are the benchmarks of [14]. As an example of an AI hierarchy, the algorithms were applied to the biological taxonomy of Mammalia (see [16], after correcting for the inconsistent indentation.)

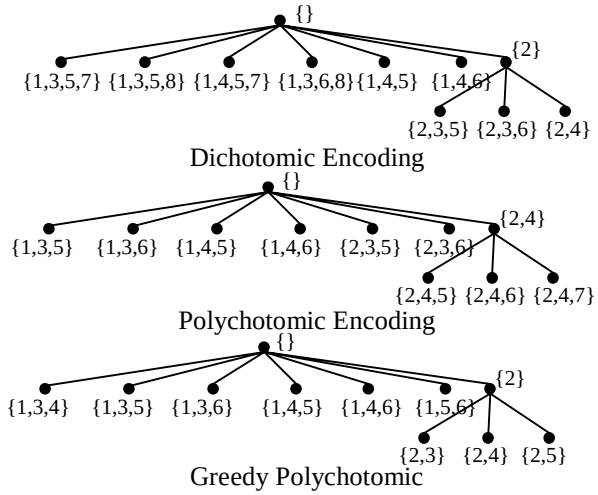

Fig. 6. Alternative encodings

	Nodes	INodes	MaxB	MeanB	SDB	MaxH	MeanH	SDH
Program classes								
NextStep	311	65	142	4.76	17.578	7	2.93	1.156
ET	371	82	87	4.51	11.131	8	3.29	1.935
Digitalk3	1357	434	142	3.12	9.873	13	5.39	2.190
Java 1.3 classes	1478	307	572	4.81	32.804	7	2.47	1.502
VisualWorks2	1957	625	181	3.12	10.400	15	6.39	2.905
Real classes								
Mammals	6059	1431	151	4.23	8.682	7	4.99	1.130
Databases								
Darwin	66991	1770	529	37.84	27.306	4	3.97	0.170
File directories								
Java 1.3 directory	1980	99	173	19.98	27.962	8	4.85	1.016
010725 C Drive	33130	1707	3139	19.40	122.338	13	6.01	2.021
010725 F Drive	114598	7286	1724	15.72	54.455	15	7.18	1.865
Complete trees								
Depth 4, width 6	1555	259	6	6.00	0.000	4	3.80	0.486
Depth 6, width 4	5461	1365	4	4.00	0.000	6	5.66	0.664

Key:: Nodes = Nodes in tree. INodes = interior (non-leaf) nodes. MaxB = maximum branching factor. MeanB = average Inode branching factor. SDB = standard deviation, branching factor. MaxH = maximum height. MeanH = mean height. STD = standard deviation, heights.

Table 2. Tree Parameters

As an example of a database system, we used the DARWIN wind-tunnel system [15]. Wind tunnel test data are hierarchical. The set of measurements about a model are a *test*. A given test can be checked for different *configurations* (e.g., orientation of the model or arrangements of sensors), a given configuration can be checked for a specific *run*, and for a run, data is collected at *points* (temporal instants). Thus, tests contain configurations which contain runs which contain points. Rather than having a single table of point data, where each line is a (fairly redundant) quadruple, the database stores this information in separate tables for each kind of data. Because access control is also hierarchical, it can be important to quickly determine if a given data element is a piece of something to which a user has access. A bit-vector encoding, at each data point, could be used to vet access.

As examples of other "naturally occurring" "computational" hierarchies, the algorithms were run on the file structure trees of the Java 1.3 distribution, and the file/directory trees of my C and F drives on July 25, 2001. I also considered two "Complete trees," one of branching factor 4 and depth 6, and other of branching factor 6 and depth 4.

Table 2 presents the structural statistics on these trees; Table 3 the number of bits each of the four algorithms requires for that tree. These examples suggest that Polychotomic Encoding is almost always better than Dichotomic Encoding, though how much better can vary considerably.

Running PE and DE on random trees enabled a better understanding of the algorithms' relative strengths.[1] The "jar graph" of Figure 7 shows the results of running DE and PE on such random trees. For each experimental point, 30 trees were generated, $\mathcal{D}$ and $\mathcal{P}$ calculated and averaged. The experiment was run with tree sizes of 200, 2,000, 20,000, and 200,000; mean branching factors of 2, 8, 32, and 128 nodes, and standard deviations that varied from one half the mean (the narrow jars), the mean (the medium jars) and twice the mean (the fat jars). The upper number on each jar is the average number of bits needed for Dichotomic Encoding; the lower number, the average number of bits needed for Polychotomic Encoding. Visually, each jar is as "full" as the ratio of these two numbers.

The increasing white space in the jars as one moves to the right suggests that PE has greater advantage when dealing with trees with higher branching factors, that this advantage is relatively independent of the tree size, and that greater variance in the branching factor produces only a minor boost to the performance of PE, perhaps as wider nodes are more likely to be in the tree.

[1] For this experiment, the random tree generator $\mathcal{G}$ was parameterized by the number of nodes to be generated, a mean branching factor, and a standard deviation of the branching factor. ($\mathcal{G}(n, m, sd)$). When $n = 0$ the algorithm would just generate a leaf node. Otherwise, a Gaussian random number generator would be repeatedly invoked with the given mean and standard deviation until it returned a positive number, b. If $b \leq n$, the node received n leaf children. If not, the allocation of $n - b$ nodes was randomly divided among the children by picking dividing points at random, and $\mathcal{G}$ recursively invoked to generate the child trees.

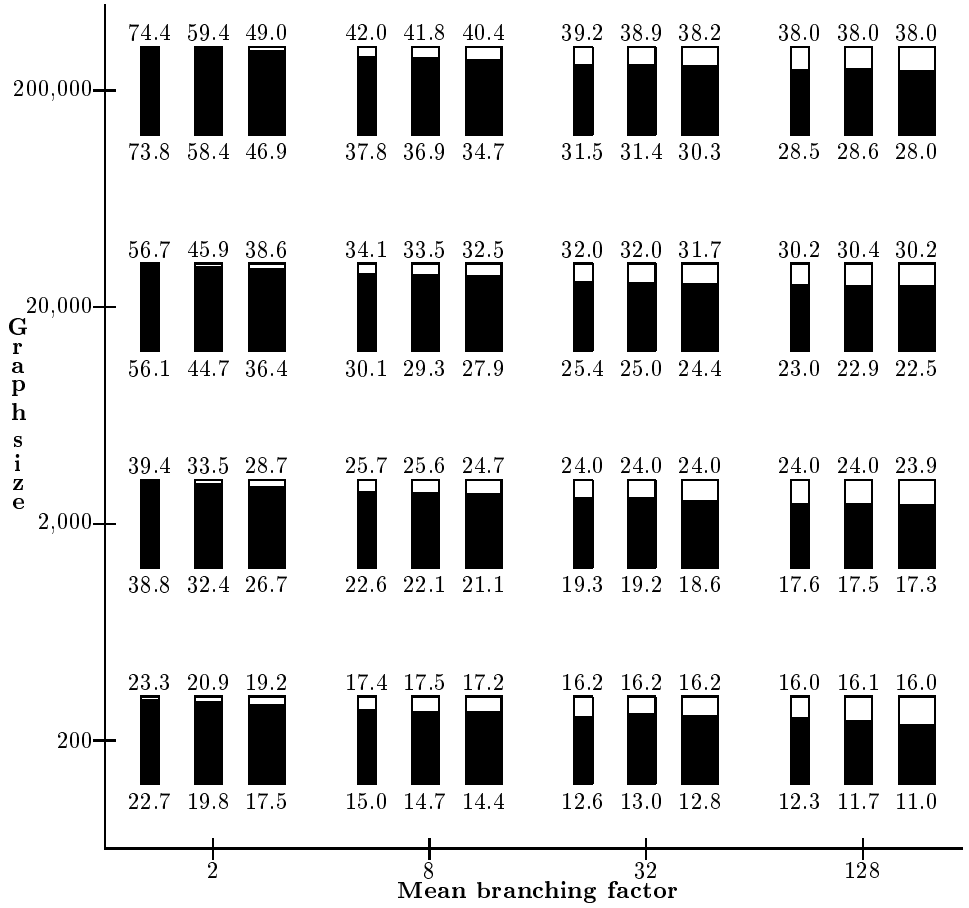

Fig. 7. Dichotomic and Polychotomic performance on random trees. Each bar shows the portion of the number of bits required by DE used by PE. The three jars in each entry show the test run with different standard deviations of the branching factor (half the mean, the mean, and twice the mean.)

	CHNR	DE	PE	GPE
Program classes				
Java 1.3 classes	36	23	21	21
Digitalk3	52	29	28	30
ET	41	20	20	22
NextStep	28	20	19	18
VisualWorks2	58	33	31	35
Real classes				
Mammals	41	30	26	29
Databases				
Darwin	30	36	28	27
File directories				
Java 1.3 directory	40	27	23	23
010725 C Drive	60	38	33	36
010725 F Drive	73	43	38	40
Complete trees				
Depth 6, width 4	24	24	24	24
Depth 4, width 6	16	24	16	16

Key:: CHNR = Multi-bit encoding. DE = Dichotomic Encoding. PE = Polychotomic Encoding. GPE = Greedy Polychotomic Encoding.

Table 3. Algorithm results

11 Discussion

This paper described the Polychotomic Encoding algorithm, an improvement to the Dichotomic Encoding algorithm of Raynaud and Thierry. Polychotomic Encoding, like its predecessor, is near-linear in execution and produces good (but not optimal) bit-vector encodings. (The problem of taking an arbitrary directed acyclic graph finding the minimal encoding is NP-hard [9]. The difficulty of trees is an open question.) An open research question suggested by this work is how best to apply hierarchical encoding algorithms to multiple inheritance graphs. A promising direction is to partition the hierarchy into slices, each of which could be quickly encoded with a different bit-set. The recent work of Zibin and Gil on PQ-trees [17] suggests a possible approach to the partitioning problem.

Acknowledgments

My thanks to Cecilia Aragon, Diana Lee, Barry Leiner, Tarang Patel, Olivier Raynaud, Eric Thierry, and Rajkumar Thirumalainambi for discussions and comments on the drafts of this paper, and to Louise Chan for help with the DARWIN database.

References

1. Aït-Kaci, H., Boyer, R., Lincoln, P., and Nasr, R.. Efficient implementation of lattice operations. *TOPLAS 11*, 1 (Jan. 1989), 115–146.
2. Baker, H. A Decision Procedure for Common Lisp's SUBTYPEP Predicate. *J. Lisp and Symbolic Computation 5*, (Sept. 1992), 157–190.
3. Caseau, Y. Efficient handling of multiple inheritance hierarchies. In *Proc. OOPSLA-93*, (Washington, D.C., 1993), 271–287.
4. Caseau, Y., Habib, M., Nourine, L., and Raynaud, O. Encoding of multiple inheritance hierarchies and partial orders. *Computational Intelligence 15*, 1 (1999), 50–62.
5. Cohen, N. H. Type-extension tests can be performed in constant time. *TOPLAS 13* (1991), 626–629.
6. Fikes, R., and Kehler, T. The role of frame-based representation in reasoning. *CACM, 28*, 9 (September 1985), 904–920.
7. Goldberg, A., and Robson, D. *SmallTalk-80: The Language and its Implementation.* Addison-Wesley, Reading, MA. (1980).
8. Gosling, J., Joy, B., Steele, G., and Bracha, G. *The Java Language Specification, 2nd Edition.* Addison-Wesley, Reading, MA, (2000).
9. Habib, M., and Nourine, L., Bit-vector encoding for partially ordered sets, *ORDAL'94*, LNCS No 831, Springer-Verlag, Berlin (1994), 1–12.
10. Keene, S. E. *Object-Oriented Programming in Common Lisp: A Programmer's Guide to CLOS.* Addison-Wesley, Reading, Massachusetts (1989).
11. Raynaud, O. and Thierry, E. A quasi optimal bit-vector encoding of tree hierarchies. Application to efficient type inclusion tests. In *Proc. ECOOP 2001.* LNCS 2072, Berlin: Springer-Verlag, (2001), 165–180.
12. Schubert, L. K., Papalaskaris, M. A., and Taugher, L. Determining type, part, color, and time relationships. *IEEE Computer 16*, 10 (October 1983), 53–60.
13. Stickel, M. E. Automatic deduction by theory resolution. In *Proc. IJCAI-85.* Los Angeles, Morgan Kauffman, Los Altos, CA (1985), 1181–1186.
14. Vitek, J., Horspool, R. N., and Krall, A. Efficient type inclusion tests. In *Proc. OOPSLA-97*, Atlanta, (October 1997), 142–157.
15. Walton, J., Filman, R. E., and Korsmeyer, D. J. The evolution of the DARWIN system. In *Proc. ACM Symposium on Applied Computing*, Como, Italy, (March 2000), 971–977.
16. Wilson, D. E., and Reeder, D. M. (Eds.). *Mammal Species of the World.* Smithsonian Institution Press, Washington, D.C., 1993. gopher://nmnhgoph.si.edu/00/.docs/mammals_data/list
17. Zibin, Y. and Gil, J. Efficient subtyping tests with PQ-Encoding. In *Proc. OOPSLA-2001*, Tampa, Florida (October 2001), 96–106.