ISSN 0254-2757

QI QUÆSTIONES INFORMATICÆ

Volume 6 • Number 2 September 1988

The official journal of the Computer Society of South Africa and of the South African Institute of Computer Scientists

Die amptelike vaktydskrif van die Rekenaarvereniging van Suid-Afrika en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

QUÆSTIONES INFORMATICÆ

The official journal of the Computer Society of South Africa and of the South African Institute of Computer Scientists

Die amptelike vaktydskrif van die Rekenaarvereniging van Suid-Afrika en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

Subscriptions

The annual subscription is

	SA	US	UK
Individuals	R20	$ 7	£ 5
Institutions	R30	$14	£10

to be sent to:
Computer Society of South Africa
Box 1714 Halfway House 1685

Quæstiones Informaticæ is prepared by the Computer Science Department of the University of the Witwatersrand and printed by Printed Matter, for the Computer Society of South Africa and the South African Institute of Computer Scientists.

Traversable Trees and Forests

S W Postma

Department of Computer Science, University of Natal, Pietermaritzburg, 3200

Abstract

Two topics are studied, related and generalised in this paper – the Knuth transform of an arbitrary tree to a binary tree, and Pfaltz's definition of a data structure as a graph with assignments. Trees are defined in terms of undirected graphs, and the binary tree is shown to be a data structure. And/or graphs are considered and generalised to fans which are shown to be Knuth transformable. A (presumably most) general structure which is Knuth transformable is then defined, a possible notation is suggested, and its implementation in Octolisp is indicated.

Computing Review Categories: *E.1 DATA STRUCTURES, G.2.2 GRAPH THEORY*

Keywords: *Design, Theory, Languages, Undirected trees, abstract Knuth transform, fan structures, tree based data structures, Octolisp.*

Received October 1987, Accepted January 1988

1. Graphs, Trees and Data Structures

In this section we review some standard definitions and details may be found in Knuth [1] or Pfaltz [3].

Trees considered as data structures are common informatic objects, so common that we tend to forget the assumptions that we make about them. In particular we tend to consider only directed rooted structures although more general cases are considered in the literature [1],[3]. The basic ideas are developed in terms of graph theory in this section.

A **graph** is a set Q of points, called the **nodes** together with a set E of **edges** where

$E = \{\{x,y\} \mid x \in Q \ \& \ y \in Q\}$

Nodes are designated by node labels q0, q1,..., qn, and the edge {qi, qj} by qiqj or by qjqi. If qiqj∈E then qi and qj are said to be **adjacent**.

Adjacency is just a symmetric relation on the set Q. In informatics, adjacency is often considered to be a (two-way) access path.

From adjacency we can derive the transitive **accessibility** relation: qj is said to be accessible from qi if there is a sequence of nodes

qi = q0, q1,..., qn = qj

such that qs ≠ qt if s ≠ t and also qs and qs+1 are adjacent for $0 \le s \le n-1$. Such a sequence of nodes is called a **path** from qi to qj. Obviously qn, ..., q0 is a path from qj to qi.

From the adjacency relation we also obtain the adjacency function $A: Q \to 2^Q: qi \to \{qs \mid qiqs \in E\}$. This function is to be used in other sections of this paper.

A graph is said to be **connected** if for any two nodes, qi and qj, there is a path from qi to qj. A connected graph is said to be a **free tree** if there is at most one path between any two nodes, i.e. a free tree is a connected graph with no cycles.

A **rooted tree**, or a tree, is a free tree in which one node is distinguished and is called the root of the tree. It is obvious that at most |Q| rooted trees are obtainable from a given free tree with Q as nodeset.

A node qi, other than the root node, of a rooted tree is called a **leaf**, if |A(qi)| = 1.

In (rooted) trees we are interested in a particular restriction of the accessibility relation – we are interested in paths from the root node to the other nodes and to the leaves in particular. We therefore define the following.

We look at two sets of definitions of direct descendants and the direct descendancy function – the first is favoured by Phillips [4] and is as follows.

Let q0 be the root of a tree, then qj is a direct descendant of qi iff there is a path q0, ..., qi, qj, and qj is a descendant of qi if there is a path q0, ..., qi, ..., qj. Furthermore D(qi) = {qj|qj is a direct descendant of qi}. The other definitions are: the direct descendancy function, $D: Q \to 2^Q$ is defined inductively by:

D(q0) = A(q0) where q0 is the root node,
at step i let

$fi = \{qs \mid D(qs) \text{ is established}\}$

$ni = \cup s\{qi \mid qi \in D(qs), qs \in fi\} \setminus (\{qj \mid qj \text{ is a leaf}\} \cup fi)$

If ni is empty then D is fully established,
else take any qk∈ ni and then D(qk) = A{qk}\fi.

In a rooted tree qj is said to be a **direct descendant** of qi if qj ∈ D(qi), and qk is said to be a **descendant** of qi if there is a sequence of nodes qi = q0, qi,..., qn = qk such that qk+1 is a direct descendant of qk. We note that every leaf is a descendant of the root node.

Instead of defining our graphs in terms of edges {x, y}, we could have defined them in terms of arcs

<x,y> such that <x,y> is an arc iff <y,x> is an arc. Each edge from E may be replaced by a pair of arcs from Ea. This point of view is now taken of the rooted trees, and the descendancy function D is used to partition the set of arcs into two disjoint sets, called Ed and Eu, as follows

$Ed = \{\langle x,y \rangle \mid y \in D(x)\}$

$Eu = \{\langle y,x \rangle \mid y \in D(x)\}$

i.e. Eu = Ea\Ed

A tree then, which may be considered to be a system (Q; Ea; q0) is now considered to be the system (Q; Ed, Eu; q0). We note that the usual directed tree as a system (Q; Ed, { }; q0).

We have to consider Pfaltz's definition of a data structure. He defines a data structure as a graph with assignments to the nodes and/or the edges. A **tree data structure** is now provisionally defined to be a system

[(Q; Ed, Eu; q0); Fq; Fe]

where Fq: $Q \to \{X \mid X$ is a node value$\}$

Fe: $Ea \to \{Y \mid Y$ is an edge value$\}$

and Fe is typically defined by cases to be

Fd: $Ed \to \{Yd \mid Yd$ is a down-edge value$\}$

Fu: $Eu \to \{Yu \mid Yu$ is an up-edge value$\}$

Hence a data structure is the system

[(Q; Ed, Eu; q0); Fq; Fd, Fu].

A more general definition of a data structure on a tree is to be developed in a later section. We note at this stage merely that in a data structure implementation, an arc is represented by a traversal access, and the value associated with an arc or node by a retrieval access.

We conclude this section by noting that a **forest** is obtained when the root node is omitted and also all arcs to and from the root node.

2. Knuth Transforms and Ordered Trees

Pfaltz [3] defines a Knuth transform to be a mapping of an n-ary tree to a "Knuth binary tree" that "preserves features of interest." Now a Knuth binary tree is not a tree but is a data structure since Knuth defines such a binary tree to be "a finite set of nodes which is either empty, or consists of a root and two disjoint binary trees called the left and the right subtrees of the root."

Two problems are raised: first, how do we characterise the "features of interest", and secondly, the use of the words "left" and "right" is just an instance of positional ordering of descendants of a node. The question is, how do we introduce positional order into our trees? Although the final results stated in section 6 resolve both problems simultaneously, we will approach the results by steps in this and the sections in between.

The abstract discussion is illustrated by examples, and we start off by considering the rooted tree

(Q = {a b c d e f g h i j k l m n o}; E = {ab ac ad ae cf cg dh di dj hk hl lm ln lo}; a)

which is represented graphically by

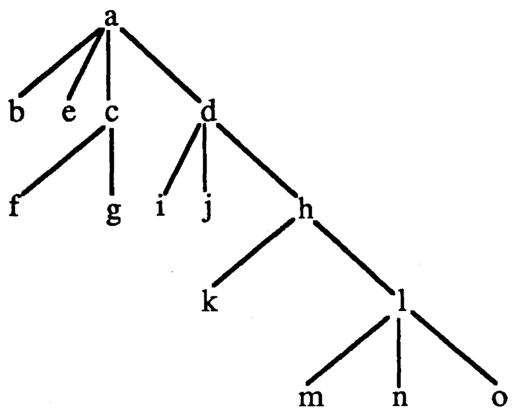

Figure 1

A possible Knuth transform of this given tree is

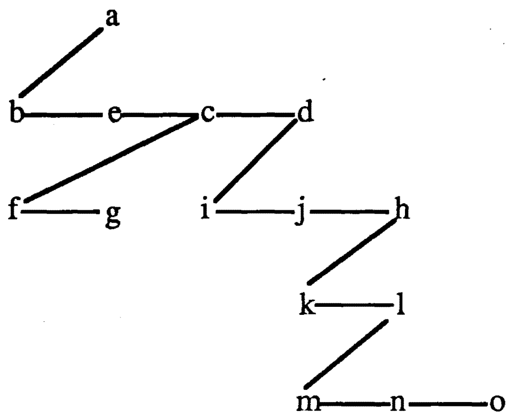

Figure 2

It may be noted that the following features of interest are preserved:

1. Descendancy
2. Order – although it is a spurious order in the representation.

In our representation we represented D(a) by the list (b e c d) instead of a set. We are thus led to define the **abstract Knuth transform** of a tree to be given by the descendancy function (i.e. D) written in the form

node : set of direct descendants

thus obtaining for the given tree

a: {b c: {f g} d: {h: {k l: {m n o} i j} e}

where the elements of each set in the transform are not positionally ordered.

An order may be imposed on a set, and the notation (u v w ...) will be used for the set {u v w ...} with an imposed order u<v, v<w, w<... . In the tree above we may have, say, that o<n<m and also j<i<h, and we see that this is easily represented in our abstract Knuth transform as follows

a: {b c: {f g} d: (j i h: {k l: (o n m)})e}

This is however, the transform of the object below, which is not a tree

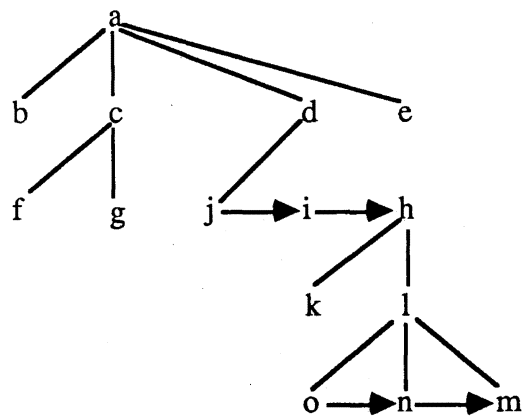

Figure 3

The object is however definable as a graph if we use arcs. We conclude that the abstract (generalised) Knuth transform is useful for mapping more general structures than trees, and will solve the characterisation problem for these structures in section 6.

3. Data Structures on Trees

The provisional definition for a data structure on a tree is given in section 1, namely that it is a system

[(Q; Ea; q0); Fq; Fe]

may be illustrated by the following example:

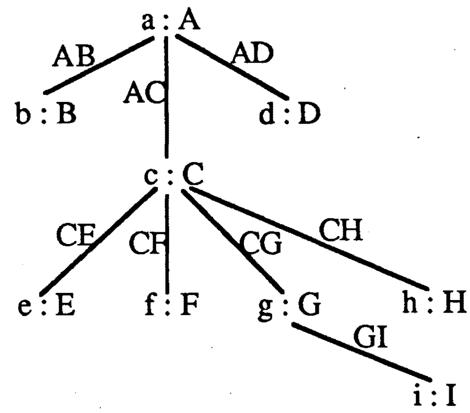

Figure 4

where Q = {a b c d e f g h i}

Fq:Q→{A B C ... I} as indicated

Fe:QxQ→{XY | xy ∈ Ea}

In a data structure we typically leave out the node labels, and obtain the tree represented graphically by

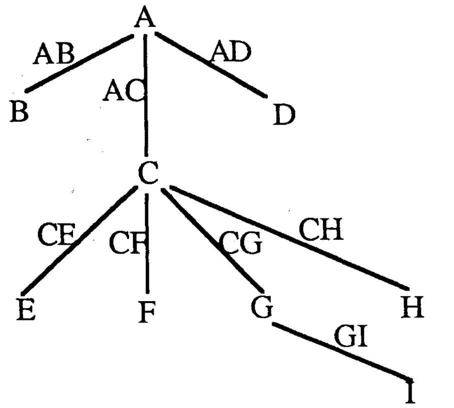

Figure 5

We now note that we can 'drop' the edge values to the direct descendants to obtain the representation (using '%' for 'no value')

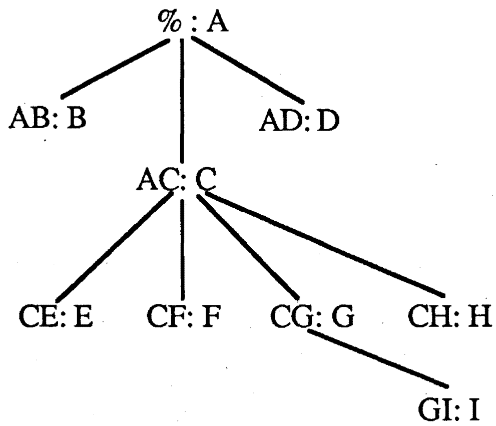

Figure 6

Hence we modify the provisional definition of a data structure on a tree to a definition as follows.

A basic tree data structure is a system

[(Q; Ea; q0); F]

where F:Q → Edge-values x Node-values

The abstract Knuth transform of the final version of the data structure is

%: A: {AB: B AC: C: {CE: E CF: F CG: G: {GI: I} CH: H} AD: D}

and we note that we have to add a constraint in that no two nodes that are siblings (i.e. direct descendants of the same parent node) may be assigned the same (pair of) values.

4. Fans and their Data Structures

A fan, as defined in this section, is a graph that is abstract Knuth transformable. The ordered trees considered in a previous section, and the AND/OR graphs of artificial intelligence [2] are shown to be data structures on fans.

Consider the following example of a fan

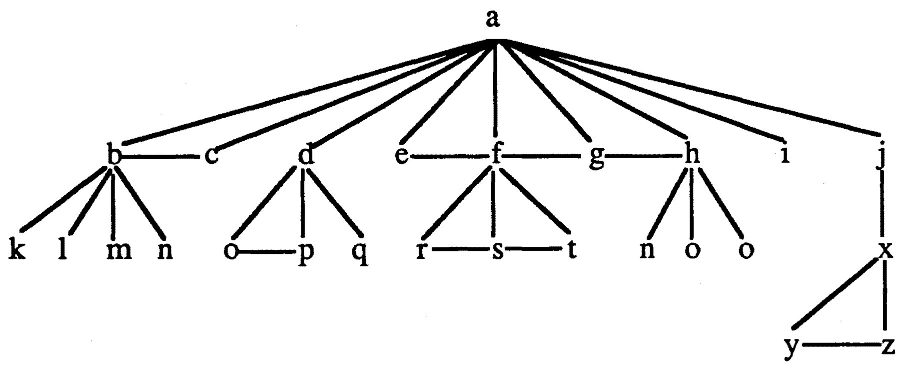

Figure 7

A **fan** is a system
(Q; Ed, Eu, Er; q0)
such that (Q; Ed, Eu; q0) is a tree;
and xy ∈ Er implies x and y are siblings;
and Er is a forest of simple paths.

If any interior node of a fan is considered, say qi, then we see that its set of direct descendants, D(qi), may be partitioned into subsets that are connected, and these are denoted by R(qi). For the fan illustrated above we have, e.g.

R(a) = {{b c} {d} {e f g h} {i} {j}}

We are now tempted to define a generalised abstract Knuth transform by taking the abstract Knuth transform of the underlying tree, and in addition enclosing every element of each R(qi) in brackets – for the sake of simplicity brackets are omitted around single elements. For our example we would obtain

1. Abstract Knuth transform of the underlying tree:
a: {b: {k l m n} c d: {o p q} e f: {r s t} g h: {u v w} i j: {x: {y z}}}
and

2. Generalised abstract Knuth transform of the fan:
a: {{h: {k l m n} c} d: {{o p} q} {e f: {{r s t}} g h: {u v w} } i j: {x: {{y z}}}}

It now seems easy to define data structures on fans. For example, the edges defined by Er could be taken to be AND relations in the fan is considered as an AND/OR graph. Or we could consider the relation of positional ordering, and for orders defined by: b<c, e<f<g<h, o<p, r<s<t and y<z we would have:

2.' a:{ (b: {k l m n} c) d: {(o p) q} (e f: {(r s t)} g h: {u v w}) i j: {x: {(y z)}}}
which would have the obvious and simplified form (q1 ... qn) for {(q1 ... qn)}.

Two observations on fans justifies the definitions to be given in the next section. First, we note that in the generalised abstract Knuth transform the term, e.g. {e f: ... g h: ...} is used, and the set notation does not convey the information that we are dealing with edges ef, fg and gh. Secondly, it is clear from AND/OR graphs that the sub-fan representation a: {...{e f g h} ...} is actually correct; considered as a grouping the relationships should be represented by the full graph {ef eg eh fg fh gh} (cf. [5]). We may say that what we actually have is that the pictorial representation of the fan should be considered to be a 'name' in the same way that {a b c} is the 'name' of the sets {b a c} or {a a b b c} etc., but a better solution is at hand.

5. Classification Mappings

The classification mappings defined in this section are used in the next section to define data structures on trees, such data structures to be generally abstract Knuth transformable.

Consider an arbitrary set, say {s1 s2 ... sn}, and define a **classification domain** for that set as follows.

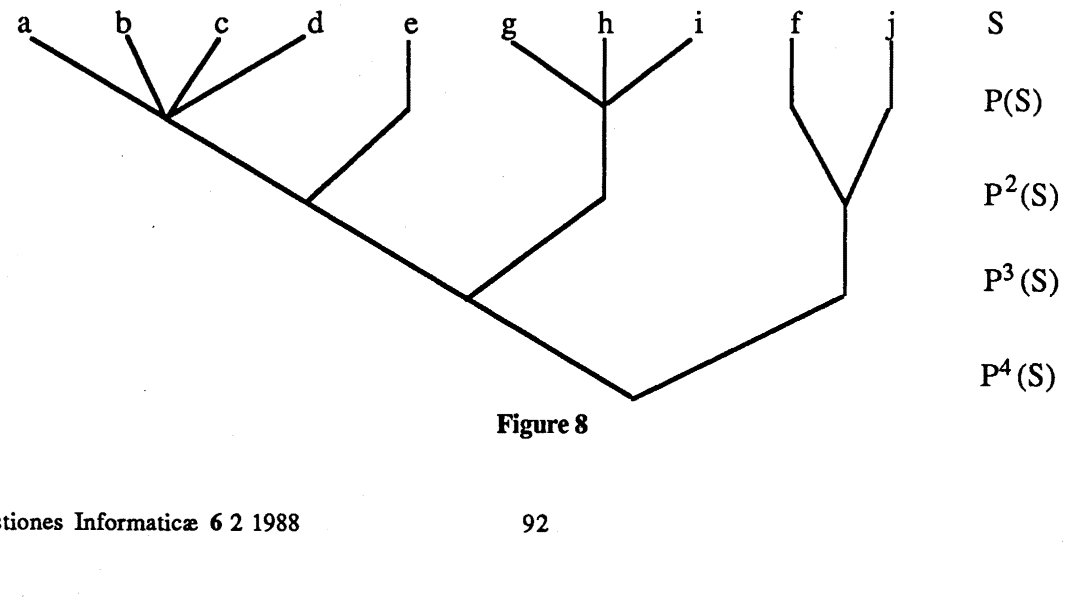

Figure 8

Let $P(S) = \{S1\ S2 \ldots Sk\}$
such that $(\forall\ si \in S)\ (\exists\ Sj)\ (si \in Sj)$,
and the Si are pairwise disjoint
and k<n.
P(S) is a partition of S.
If all the conditions cannot be met then P(S) = { }.
If P{S} is not empty then it is a set and we can find a P(P(S)), hence we define C(S), the classification domain to be $\cup_i P^i(S)$ where $P^i(S) = P(P^{i-1}(S))$.

For example, let S = {a b c d e f g h i j}
Then one classification domain for S is:
$P(S) = \{\{a\,b\,c\,d\}\ \{e\}\ \{f\}\ \{g\,h\,i\}\ \{j\}\}$
$P^2(S) = \{\{\{abcd\}\ \{e\}\}\ \{\{f\}\ \{j\}\}\ \{\{g\,h\,i\}\}\}$
$P^3(S) = \{\{\ \{\{abcd\}\ \{e\}\}\ \{\{ghi\}\}\ \}\ \{\ \{\{f\}\ \{j\}\}\ \}\ \}$
$P^4(S) = \{\{\ \{\{\{abcd\}\ \{e\}\}\ \{\{ghi\}\}\}\ \{\{\{f\}\ \{j\}\}\ \}\ \}\ \}$
and $C(S) = P(S) \cup P^2(S) \cup P^3(S) \cup P^4(S)$.

Each classification domain has an associated tree, and for the domain of the example we have the diagram shown in Figure 8.

A **classification mapping** on a set is a mapping
F: $C(S) \rightarrow V$ defined by cases on the $P^i(S)$ subsets of C(S). That is
$F: P(S) \rightarrow V1$
$P^2(S) \rightarrow V2$
.
.
.
$P^i(S) \rightarrow Vi$ where $P^{i+1}(S) = \{\ \}$
and $V1 \cup V2 \cup \ldots\ldots Vi \subseteq V$

6. Generalised Tree Based Data Structures

This section contains the final results of the papaer.

It was shown in a preceding section that it is sufficient to consider only mappings to nodes in defining a data structure, and hence we define:

A **tree based data structure** is a basic tree data structure together with a classification mapping for the root node and each set of siblings. The definition is easily extended to forests.

A **forest based data structure** is a forest of basic tree data structures together with a classification mapping for each set of siblings and a classification mapping for the set of root nodes.

Since each classification mapping has a corresponding tree it is obvious that we can find an abstract Knuth transform for the tree and if we add information about the classification mapping we call it an **extended Knuth transform** which is defined below.

Our range sets V0, used for the basic tree data structures, are actually product sets in that they may contain node-values and/or edge-values. In our extended tranformation we use the convention:
node-value:
edge-value: –
edge-value: – node-value:
i.e. node-value: corresponds to %: – node-value:
and edge-value: – corresponds to edge-value: –%:

Two cases occur as values of the Vi, i>1, sets with such regularity that a special notation is justified. These are the cases where positional ordering is specified, or the set-like unordered structure is maintained. When ordering is specified, e.g. the element {A B C D} is mapped to {<A, B>, <B, D>, <D,C>} it is indicated by parentheses, e.g. (A B D C}, otherwise square brackets are used, i.e. [A B C D]. The element {A B C D} may also be mapped to some other value, say X, in which case that value is shown as follows:
unordered case [: =X A B C D]
ordered case (: =X A B C D)
i.e. (A B D C)
is [: ={<A,B>, <B, D>, <D,C>} A B C D]
or is [: =X5 A B C D]
where X5 = {<A,B>, <B, D>, <D, C>}

An example is considered before the abstract formulation is finalised.

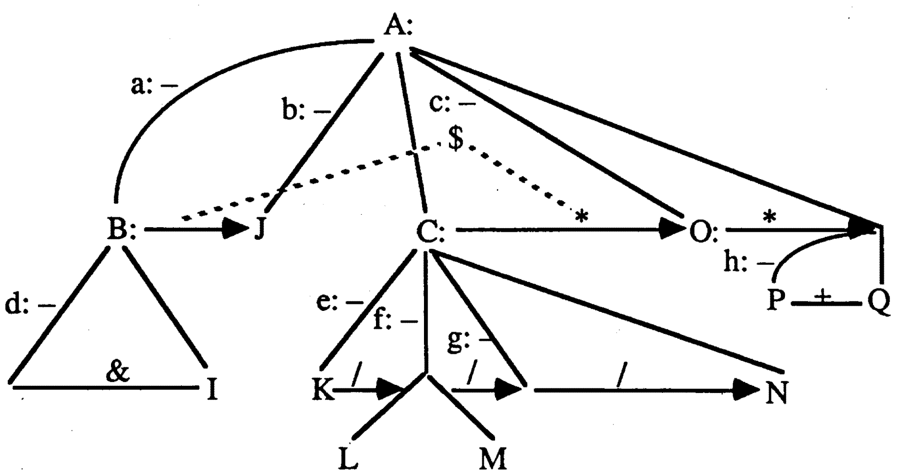

Figure 9

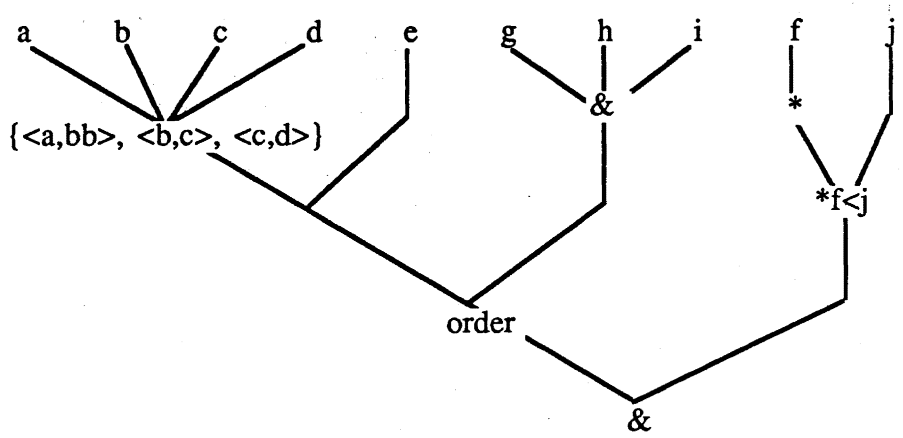

Figure 10

which is to have the extended transform:

A: {: =$ (a: –B: [:=& d: –% I] b: –J) (: =* C: (: =/ e: –K f: –[L M] g: –% N) c: –O [: =+ h: –P Q])]

Let us consider the example of the tree associated with a classification mapping from the previous section again, but with the values from the Vi shown at the nodes. We get, e.g Figure 10.

If we take a modified Knuth transform, i.e. node value written following, not preceding. the open brackets, we get:

[: =& [: =order [: =% [: ={<ab>...} a b c d] [: =% e]] [: =% [: =& g h i]]] [: =% [: =order [: =* f] [: =% j]]]]

The whole set {a b ... j} is a set of siblings, descendants of say, of a node with value A, and we write

A: [: =& ...]. Each of the a, ..., j represents a subtree and its transform may be substituted for the given nodes.

7. Conclusion: Applications

Three aspects of applications of the theory expounded are considered in this section: implementation in a program language, models for ADT's (abstract data types) and program specification.

The current version of Quadlisp, QL/86, is being revised to obtain Octolisp. As a part of the revision the trees (and forests) described in this paper are implemented with the restriction that an interior node must have an edge-value or a node-value but not both, and the notation:

{ ... } for		[...]
{ ... } for	[: =& ...]	conjunction
{ ... } for	[: =/ ...]	disjunction
{ ... } for		(...) ordered
{ ... } for	(: =& ...)	ordered conjunction
{ ... } for	(: =/ ...)	ordered disjunction
{ ... } for	[: =~ ...]	negation

In all cases the additional specification : =val, e.g. {: =Sigma ... } is allowed.

Furthermore the trees or forests may at a sibling level be considered to be streams.

A second application for these generalised tree data structures is to set up models for ADT's defined for tree or forest structures. Since the generalised tree data structures are mathematical objects this will solve existence problems constructively.

Finally, in program specification we need concepts that are as general as possible. We surmise that the objects defined in this paper are the most general objects that have abstract Knuth transforms.

As an open problem we leave the (mathematical) investigation of the possibility of defining classification mappings on the trees associated with classification mappings. We surmise that such structures may be useful in semantics and program verification.

References

[1] D.E. Knuth, [1968], *The Art of Computer Programming – Vol 1, Fundamental Algorithms*, Addison Wesley, Reading Mass.

[2] N.J. Nilsson, [1971], *Problem Solving Methods in Artificial Intelligence*, McGraw Hill, New York.

[3] J.L. Pfaltz, [1977], *Computer Data Structures*, McGraw-Hill, New York.

[4] N.C.K. Phillips, [1987], personal communication.

[5] S. Postma, [1982], On the definition and implementation of the programming language Quadlisp, Ph D Thesis, UNISA, Pretoria.

NOTES FOR CONTRIBUTORS

The purpose of the journal will be to publish original papers in any field of computing. Papers submitted may be research articles, review articles and exploratory articles of general interest to readers of the journal. The preferred languages of the journal will be the congress languages of IFIP although papers in other languages will not be precluded.

Manuscripts should be submitted in triplicate to:

Professor J M Bishop
Department of Computer Science
University of the Witwatersrand
Johannesburg
Wits
2050

Form of manuscript

Manuscripts should be in double-space typing on one side only of sheets of A4 size with wide margins.

The first page should include the article title (which should be brief), the author's name and affiliation and address. Each paper must be accompanied by an abstract less than 200 words which will be printed at the beginning of the paper, together with an appropriate key word list and a list of relevant Computing Review categories.

Manuscripts may be provided on disc using any Apple Macintosh package or in ASCII format.

For authors wishing to provide camera-ready copy, a page specification is freely available on request from the Editor.

Tables and figures

Tables and figures should not be included in the text, although tables and figures should be referred to in the printed text. Tables should be typed on separate sheets and should be numbered consecutively and titled.

Figures should also be supplied on separate sheets, and each should be clearly identified on the back in pencil and the author's name and figure number. Original line drawings (not photocopies) should be submitted and should include all the relevant details. Photographs used as illustrations should be avoided if possible. If this cannot be avoided, glossy bromide prints are required.

Symbols

Mathematical and other symbols may be either handwritten or typewritten. Greek letters and unusual symbols should be identified in the margin. Distinction should be made between capital and lower case letters; between the letter O and zero; between the letter I, the number one and prime; between K and kappa.

References

References should be listed at the end of the manuscript in alphabetic order of the author's name, and cited in the text in square brackets. Journal references should be arranged thus:

[1] E. Ashcroft and Z. Manna, [1972], The Translation of 'GOTO' Programs to 'WHILE' programs, *Proceedings of IFIP Congress 71*, North-Holland, Amsterdam, 250-255.

[2] C. Bohm and G. Jacopini, [1966], Flow Diagrams, Turing Machines and Languages with only Two Formation Rules, *Comm. ACM*, **9**, 366-371.

[3] S. Ginsburg, [1966], *Mathematical Theory of Context-free Languages* McGraw Hill, NewYork.

Proofs

Proofs will be sent to the author to ensure that the papers have been correctly typeset and *not* for the addition of new material or major amendment to the texts. Excessive alterations may be disallowed. Corrected proofs must be returned to the production manager within three days to minimise the risk of the author's contribution having to be held over to a later issue.

Only orginal papers will be accepted, and copyright in published papers will be vested in the publisher.

Letters

A section of "Letters to the Editor" (each limited to about 500 words) will provide a forum for discussion of recent problems