高機能アクセス法 AAM の形式モデルとその実現方式†

有澤 博†† 岡山利次†††
鈴鹿豊明††† 久保 隆†††

既存のファイルシステムには様々な問題点があるが，中でもファイルオペレーションの能力が単純で貧弱であることが，アプリケーションプログラムに負担をかける要因となってきた．本報告では高レベルのオペレーションで上位のアプリケーションの負担を軽減するという点から，ファイルオペレーションの抽象化モデル AAM (Advanced Access Method) を提案する．このファイルシステムの設計の際には抽象データ型 (ADT) の概念を用い，ADT をその上に定義されたオペレーションまで含めた数学モデルとして考えた．AAM では，扱うファイルを集合としてとらえ，その集合の要素間における「順序」の概念を基本とすることで，キーによるアクセスまでも含めた統一的なオペレーションの体系を定義する．順序の扱いについては，任意の位置に要素を入れたり，任意個の要素を飛び超すなどのオペレーションを考えており，しかもそれが要素数の対数オーダで行い得るという実現性に裏打ちされたものである．また，このオペレーション体系は，他のオペレーションの組合せで実現できないという意味で最小のオペレーション集合であるが，その組合せで既存のほとんどのファイルオペレーションを包括できる強力なものである．我々は，AAM を実現するアクセス法を実際に試作してきたが，本報告では，この AAM の試作システムに用いた一連の技法である重み付き B トリーや，AAM を用いたアプリケーションの例についても述べる．

1. まえがき

最近，パーソナルコンピュータやいわゆるワークステーションと呼ばれるクラスのコンピュータで，ハードディスクによる大容量の外部記憶装置を持つものが増えてきた．一方，これに伴って，ソフトウェアの方も大容量高速アクセスファイルの存在を前提としたものが目立つようになった．例えば，簡易型のデータベースシステムや，大容量テキストを対象としたワードプロセッサなどがこれに当たるであろう．

ところで，ユーザがファイルアクセスを必要とするアプリケーションを開発する際に，従来のファイルシステムでは「使いにくい」というのが一般的な印象である．その理由は，

（1） 言語などに比べファイルオペレーションの標準化（統一）が進んでいない．

（2） 現在のオペレーティングシステムでは，非常に低レベルのファイルオペレーションしか許していない．

などが挙げられる．本論文では，このうち(2)の観点からファイルオペレーションの実用レベルでの抽象化を行ったモデル **AAM** (Advanced Access Method) を提案し，このモデルを既存の計算機上で試作，実現した際に用いた技法について述べる．

文献1),2)においては，特にデータベースの効率の良い実現に重点を置いて関係テーブル中へのタプルの挿入，削除等のオペレーションが提案されている．しかし，本論文における著者らの態度は，さらに基本的な「ファイル」といわれるデータ集合に対して，「有用な」ADT（抽象データ型）オペレーション[3]のクラスを定義しようとするものである．ここでいう「有用な」とは，オペレーション集合自体はできるだけ「やせて」いる（すなわち，できるだけ少数のオペレーションからなる）が，これらを組み合わせてユーザが非常に「豊か」な操作系を構築できるものが良いという意味である．また，これらの基本オペレーションは，互いに直交していなければならない．すなわち，他のオペレーションの組合せで，ある別のオペレーションが実現できてはならない．ただし，ここでの「実現」は論理的に要素数に線型より小さいオーダで実現可能なものでなければならない．例えば，要素を任意の個数だけ削除するというオペレーションは，要素を一つだけ削除するというオペレーションを繰り返せば実現できるが，これはオーダが要素数に線型なので前者を基本オペレーションの一つと考えるべきである．

ファイルオペレーションは，その個々の要素のアク

† A Formal Model of Advanced Access Method (AAM) and Its Implementation by HIROSHI ARISAWA (Department of Computer Engineering, Faculty of Engineering, Yokohama National University), TOSHITSUGU OKAYAMA, TOYOAKI SUZUKA and TAKASHI KUBO (Research & Development Department, Hitachi Software Engineering Co., Ltd.).

†† 横浜国立大学工学部電子情報工学科
††† 日立ソフトウェアエンジニアリング(株)研究部

セスの仕方に関して本質的にキーによるアクセスと順序によるアクセスの二つに分かれると考えられる．キーによるアクセスとは全要素（レコード）にユニークな固定長文字列（ストリング）が与えられており，外部からこれを指定することによって任意の要素に直接的にアクセスすることである．順序によるアクセスには幾つかの考え方があるが，最も一般的なのは，集合の濃度（要素数）をnとしたとき，自然数の部分集合 $\{1, 2, \cdots, n\}$ との間に 1：1 の写像が存在すること，言い換えれば要素に 1, 2, …, n と番号付けができ，その番号によって任意の要素に直接アクセスできることである．ここで注意しておかなければならないことは，ここでの順序番号は純粋に順序を与えるためだけのものであり，それ以上の意味を持ってはいないことである．したがって例えば5番目と6番目の要素の間に新しい要素を挿入すると，前の6番目の要素は新たに「7番目の要素」と呼ばれる．

キーによるアクセスを実現したファイルアクセス法としては，大型汎用機上の VSAM[4] が汎用かつ高機能といわれている．しかし，VSAM では，ユーザがセカンダリインデクスやサブアロケーションなどの複雑な構造，オペレーションを理解しなければならず，オーバヘッドの大きさなどを考えるとパーソナルコンピュータやワークステーション向きとはいえない．

また，順序によるアクセスについては，一般のファイルシステムでは古くから「磁気テープ上に並んだレコード群を扱う」という発想を拡張したものとして存在するが，

「ファイル中の n 番目のレコードと $n+1$ 番目のレコードとの間に新たにレコードを一つ挿入する．」

といったオペレーションまで実現しているものはなく，十分なオペレーション体系を持つものはない．

一方，UNIX ソフトウェア[5]* は，非常に簡素なファイルシステムを提供している．テキストファイルは文字の順序列だけからなり，ユーザはファイルから一文字一文字読んだり，任意個の文字を飛び越したり，テキストの部分列を他の文字列で置き換えたりできる．しかし，ここでも，上述の挿入オペレーションは不可能である．

これら既存のアクセス法に対し，AAM では前述したようにファイルアクセスの本質はキーによるアクセスと順序によるアクセスの二種であるととらえ，これらを簡素で統一的に扱え，必要な機能を網羅するよう全体を体系化する．以下，2章では，AAM の基本概念と定義について述べる．3章では，AAM の有用性を示す一つの例として，AAM の上位に構築可能な複合値を許すn項関係物理データベース編成の一方式を示す．最後に，4章では，試作システムで用いている AAM オペレーションの効率の良い実現方式について，「重み付き B トリー」とこれに関する一連の新技法を示し，その考え方を述べる．

2. AAM の基本概念とオペレーション

本章では，著者らが提案する高機能アクセス法の形式的な定義を与える．まえがきでも述べたように，我我の目的は，既存のファイルアクセス法が有している様々な機能を抽象化し，また明らかに欠如している機能を付け加えて，真に有用で単純化されたファイル構造と操作系のモデルを定義することである．

2.1 集　　合

データ構造の表現には様々な方式があるが，本論文では，抽象データ型（ADT）の概念を用いて，AAM のファイルシステムを定義する．ここでは，ADT を，そのモデルの上に定義されたオペレーションまで含めた数学モデルとして考える．ADT では，オペレーションのオペランドとして，定義された ADT のインスタンスだけでなく，整数などのような他のインスタンスも取ることができる．しかし，ADT オペレーションは，ADT 自身をオペランドとして取ることはない．これは，それぞれのオペレーションが一時に注目できるのは，ただ一つの ADT（集合）であり，その「カレント」の ADT が常に定義されているからである．本論文の AAM のオペレーションは，次のように表される．

operation (operand 1, operand 2, …).

以下に AAM ファイルシステムの定義を示す．ここでは，ファイルという用語を使わず**集合**という用語を用いる．（なお，以下で述べる集合の概念は計算機科学の分野では比較的一般的なものであるが[3),6)]，純数学的な通念からは多少それている．）

集合は，**要素**の集まりであり，要素は，集合または**アトム**である．アトムは，AAM で扱うデータの最小単位である．ここでは，一つのアトムは「バイト」に対応すると思ってよい．しかし，AAM はアトムの意味には関知せず，それを，数値，文字あるいは真偽値のいずれと解釈するかなどは，ファイルシステムの外

* UNIX ソフトウェアは，米国 AT&T 社ベル研究所が開発したオペレーティングシステムの名称である．

側の問題であると考える．いずれにしてもアトムは値（記号値）を持ち，記述される．

集合もまたある種の記号値により記述されるが，アトムと異なるのは，集合がこの記号値で識別されるという点である．この記号値は，集合に対してユニークであり，変わることがない．この記号値のことを**集合ID**と呼ぶ．集合IDは，集合の生成と同時に付与される識別子であり，アトムのように外部世界と対応した意味を持っているわけではない．

要素を一つも含まない集合を**空**な集合であるという．すべての集合は，新たに作られた時点で空である．集合は，**アトム集合** (atom set)，**集合集合** (set set)，**キー付き集合集合** (keyed set set) の三つに分類される．これを**集合タイプ** (set type) と呼ぶ．AAMの集合は要素どうしの間に線型の順序を持つが，アトム集合および集合集合では，これは外部から与える任意の順序である．アトム集合と集合集合の違いは，前者の要素がアトムであるのに対して，後者の要素はアトムでなく集合である点である．このことによって，「集合の集合」，「集合の集合の集合」のような任意の階層化ができる．

一方，キー付き集合集合では各要素はユニークなキーを持ち，要素の順は自動的にキー順に設定される．ここに，キーとは有限個のアトムの列であり，キー順とはAAMが一方的に与える順序で，通常の「辞書順」と思ってもよい．キー付き集合集合では，キーの順序によるアクセスのほかに，キーの値で直接要素を識別できるが，逆に外部から任意の順序を与えることはできない．なお，一つ一つのアトムをキーで識別することにはあまり意味がないので，キー付きのアトムの集合という集合タイプは考えていない．

直観的な理解のために，AAMの集合の例を**図1**に示す．この図で，集合は楕円で表し，要素は小さい円で表す．アトムは，一文字の英字で表し，要素を示す円の中に書いてある．

2.2 ADT オペレーション

AAM集合はADTとして扱うため，そのオペレーションも完全に定義する必要がある．以下にAAM集合のADTオペレーションを示す．

各集合タイプに許されるオペレーションはそれぞれ異なるが，要素の挿入を除き，アトム集合に対するオペレーションはすべて集合集合に適用でき，集合集合に対するオペレーションはすべてキー付き集合集合に適用できる（**図2**）．ここで例外とした挿入オペレーションは，要素がアトムであるか集合であるかの違いや，キーを付けるかどうかで，集合タイプごとにその内容が必然的に異なる．

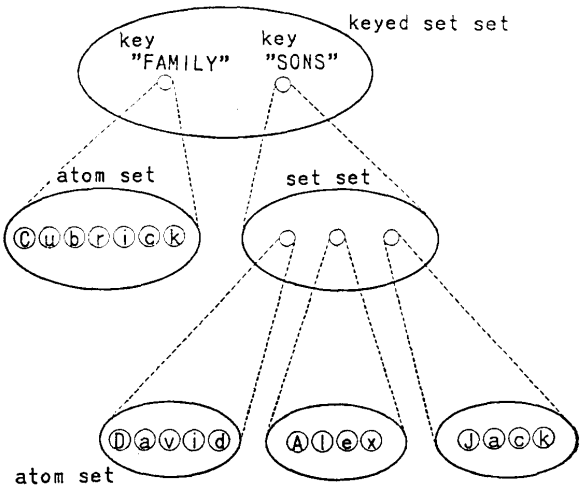

図1 AAMの集合構造の例
Fig. 1 An example of AAM set structure.

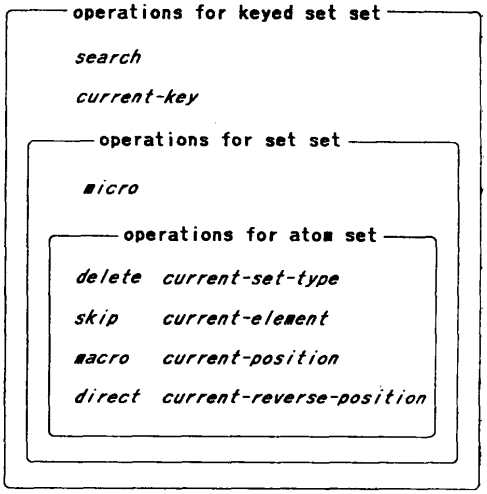

図2 AAMのオペレーション体系（挿入オペレーションを除く）
Fig. 2 Operation hierarchy for AAM (except insert-operation).

2.1節で述べたように，AAM集合は要素間に順序を持つため，要素の順をたどる「巡航」オペレーションは基本的なものであり，すべての集合タイプに共通する．特に，アトム集合と集合集合では個々の要素を指示 (point) するよりどころは，この順序しかない．我々はオペレーション体系を完成させるため，以下の説明では**カレント集合**，**カレント要素**および**ファント**ムという三つの概念を導入する．AAMでは，ある時点で注目できるのは，ただ一つの集合とその集合の中

のただ一つの要素である．前者をカレント集合，後者をカレント要素と呼ぶ．これは，一般のファイルシステムでは，「現在の位置」を示すポインタに相当する．次に，ファントムというのは集合内の要素であるが，集合とかアトムとかの実体はなく仮想的なものである．集合は新たに作成された時点で空であるが，ファントムがただ一つ存在する．このときのカレント要素はファントムである．すなわち，ファントムは空集合に対してカレント要素が「不定」となることを防ぐ働きをする．

以下に，各オペレーションの詳細について述べる．

2.2.1 更新オペレーション

更新オペレーションには挿入と削除があるが，このうち，挿入オペレーションは前述のように各集合タイプごとに異なり，異種の挿入オペレーションを適用することはできない．ここでは，各集合タイプごとの挿入オペレーションを列挙する．

初めに，アトム集合の挿入オペレーションではアトムを指定する必要がある．

(1) insert-atom (α). ここでαはアトム（を記述する記号値）である．アトムαを，カレント要素のアトムの前に挿入する．新たなカレント要素は挿入した要素となる．

これに対して，集合集合の挿入オペレーションは，挿入する要素（集合）のタイプを指定する．

(2) insert-set (τ). ここでτは集合タイプである．集合タイプτの集合を新たに作成し，その空な集合をカレント要素の前に挿入する．新たなカレント要素は挿入した要素となる．

また，キー付き集合集合の挿入オペレーションは，集合集合と同様に集合タイプを指定するが，さらに要素に付けるユニークなキーも指定しなければならない．

(3) insert-set-with-key (τ, κ). ここでτは集合タイプで，κはキーである．集合タイプτの集合を新たに作成し，その空な集合をキーκを持つ要素として挿入する．新たなカレント要素は挿入した要素となる．既に集合中に同一のキーを持つ要素が存在するときは，このオペレーションは定義されない．

これらの挿入オペレーションをどう適用しても，ファントムの後ろに要素を挿入することはできず，ファントムは常に集合の最後の要素ということになる．

次に，削除オペレーションであるが，これは挿入オペレーションと異なり，すべての集合タイプに共通である．

(4) delete (η). ここでηは整数である．ηが正数なら，カレント要素およびこれに続く$\eta-1$個の要素を削除する．新たなカレント要素は，削除した要素の後ろの要素となる．ηが負数なら，カレント要素より前の要素を$-\eta$個削除し，カレント要素は変わらない．

ファントムは，集合の生成時点から存在するもので，削除オペレーションでも削除することができない．また，削除すべき要素の数が足りないと，削除オペレーションは定義されない．

2.2.2 カレント要素変更オペレーション

以下のオペレーションはカレント要素の変更に関するものである．更新オペレーションではカレント要素は更新の結果として自然に他の要素に変わるのに対して，これらのオペレーションはカレント要素を明示的に変えるためのものである．カレント要素の変更には，順序をたどって変えるものと，キーを与えることでそのキーを持つ要素に直接位置付けるものがある．

前者は「巡航」に関するもので，アトム集合，集合集合，キー付き集合集合に共通である．

(5) skip (η). ここでηは整数である．カレント要素を元の位置から相対的にη番目の要素に変える．ηが正数なら集合の末尾方向への移動を，ηが負数なら集合の先頭方向への移動を示す．

一方，次に述べるオペレーションは，キーを扱うものであり，キー付き集合集合にだけ適用できる．

(6) search (κ) ここでκはキーである．キーκを持つ要素を探す．もし見つかれば，カレント要素をこの要素に変える．見つからなければ，カレント要素はファントムになる．

2.2.3 カレント集合変更オペレーション

図1で示したように，AAM の集合は階層的な構造をなす．したがって，異なる集合階層の任意の要素に位置付けるためには，集合から要素へ（親から子へ），あるいは，要素から集合へ（子から親へ）といったカレント集合の移動が必要となる．これは，集合内での要素間の移動が水平的であるのに対して，垂直的な移動ということができる．カレント集合の移動をともなうオペレーションを以下に挙げる．

(7) micro ( ). カレント集合を変え，元のカレント集合の，元のカレント要素を新たなカレント集合とする．新たなカレント要素は，新たな集合の先頭の要素となる．アトム集合に対しては，それ以下のレベル

の集合は存在しないので，このオペレーションは定義されない．また，元のカレント要素がファントムの場合も同様に定義されない．

(8) macro ( ). micro オペレーションの逆．すなわち，新たなカレント要素を元のカレント集合とする．新たなカレント集合は，この新たなカレント要素を含んでいる1レベル上の集合となる．

AAM システムでは，全システムを階層構造としてとらえるが，ルートという特別な集合があり，ルートにはいかなる要素からでも macro オペレーションを繰り返すことでたどりつける．すなわち，集合と要素という上下関係を通して，ルートは，すべての集合の直接，あるいは，間接の親となっている．この特別の場合として，ルート自身ルートの親であり要素でもある．このため，カレント集合がルートのときに macro オペレーションを行っても，カレント集合は変わらない．このような集合の階層は，UNIX ソフトウェアの階層ファイル構造を一般化した形になっている．

micro, macro オペレーションのように集合と要素の関係をたどるのではなく，もっと直接的に効率良くカレント集合を変えるのが次のオペレーションである．これは，明示的に集合 ID で識別される集合を探し，カレント集合をこれに変える．

(9) direct(γ). ここで，γ は集合 ID である．カレント集合を γ で識別される集合に直接変える．新たなカレント要素は，新たな集合の先頭の要素となる．

direct オペレーションを用いれば，任意の集合に直接アクセスすることができる．集合 ID をアトムの記号値で表現すれば，ファイル（集合）のインデクス，すなわちインデクスファイルを効率良く実現することができる．この例は3章で示す．

2.2.4 カレント情報オペレーション

次に挙げるオペレーションは，更新を行ったり，カレントを変更したりするものではなく，それらのオペレーションで変化した状態を示すものであり，リターン値として情報を返す．

(10) current-set-type ( ). カレント集合の集合タイプを返す．

(11) current-element ( ). カレント要素を記述する記号値を返す．すなわち，カレント要素がアトムである場合はそのアトムの値を，カレント要素が集合である場合はその集合 ID を返す．ファントムはアトムでも集合でもないため，カレント要素がファントムのときは，このオペレーションは定義されない．

(12) current-key ( ). カレント要素のキーを返す．これは，キー付き集合集合にだけ適用できる．

(13) current-position ( ). 先頭から数えた要素の位置を返す．先頭の要素の位置は常に0と数える．

(14) current-reverse-position ( ). 末尾から数えた要素の位置を返す．ファントムの位置は常に0と数える．

後で述べるように，集合内の要素の個数は，(13)のオペレーションと(14)のオペレーションの和となる．また，delete や skip オペレーションは，オペランドの η が次の条件を満たすときに定義される．

$$-\text{current-position}(\ ) \leq \eta \leq \text{current-reverse-position}(\ ).$$

2.3 AAM オペレーションの関数表記

AAM のオペレーションには，2.2.4項のように値を返すものがあるが，他のオペレーションも便宜的に空の値を返すものと考える．したがって，オペレーションをプログラミング言語における関数とみなすことができる．オペレーションのオペランドは，関数の引数に対応する．オペレーションは，対応する関数が評価されたときに実行される．それぞれの関数について，まず引数が右から左に評価され，その後に，関数自身が評価される．ここで，関数の記述方式を拡張し，関数が余分の引数を取ってもよいことにする．余分な引数は，評価されるが，関数には渡らない．

例えば，

insert-atom(α, delete(1))

では，カレント要素を削除した後に，その位置に α を入れる．すなわち，カレント要素を α に変える．これは，UNIX ソフトウェアの “putchar” 関数の機能に等しい．

2.4 プリミティブオペレーション

AAM オペレーションを設計する際の基本は，オペレーションの数を最小にすることであり，あるオペレーションが他のオペレーションや代数的オペレーションの組合せで合成できるときは，そのオペレーションは除く．このような意味で過不足ない必要最小限のオペレーションをプリミティブと呼ぶ．ただし，「要素数に線型の回数繰り返せば実現できる」というのは合成できるとはいえない．例えば，カレント要素を一つだけ移動するオペレーションを用意して skip オペレーションを用意しないのはプリミティブとして適当ではない．このような意味で AAM のプリミティブオペレーションは，上述の 14 個である．

プリミティブオペレーションの組合せで表現できるオペレーションの例として，2.3 節の putchar 関数がある．また，カレント要素を強制的に先頭およびファントムに位置付けるオペレーションは，それぞれ，

skip (-current-position ( ))

skip (current-reverse-position ( ))

で表せる．このような例は，カレント情報オペレーションにもあり，カレント集合の要素の数は，

current-position ( )+

current-reverse-position ( )

のように，AAM オペレーションの値の和になる．

このように，AAM の 14 個のオペレーションは単純で最小限の構成であるが，その組合せで既存のほとんどのファイルオペレーションを包括する強力なものである．

3. AAM の応用例—テーブルの表現方式

ここでは，AAM のアプリケーションの例としてテーブル状の構造をどのように表すか，その設計例を示す．ここで，データは，非第一正規形リレーション (NFR)[7] の形式で編成されているとする．直観的に理解しやすいように，**図 3** の例を用いて説明する．これは，文献情報を格納する小規模のデータベースである．

NFR は，二次元のテーブルであって，テーブルは複数の列からなる．それぞれの行をタプルと呼ぶが，これは n 次の述語を表し，各列は述語の役割(ロール)に対応する．このロールに付ける名前を属性と呼び，図 2 では，各列の頭の英字がこれにあたる．この例では，属性 A, T, P, Y, K は，それぞれ，「筆者」，「タイトル」，「出版社」，「出版年」，「キーワード」に対応する．例えば，一番目のタプルは，“Rex Game” という文献が，“Yas”, “Pat”, “Gch” の三名の著者によって書かれて，“Abra” 社から 1986 年に出版され，また，そのキーワードは，“DBMS” および “MODEL” であることを示す．NFR では，どの属性でも多重値を取ることができる．この結果，タプルごとのデータ

A	T	P	Y	K
Yas, Pat, Gch	Rex Game	Abra	1986	DBMS, MODEL
Ari, Yas	AIS LAN	YNU	1983	LAN, DBM

図 3 NFR の例
Fig. 3 An example of NFR (Non-First-normal-form Relation).

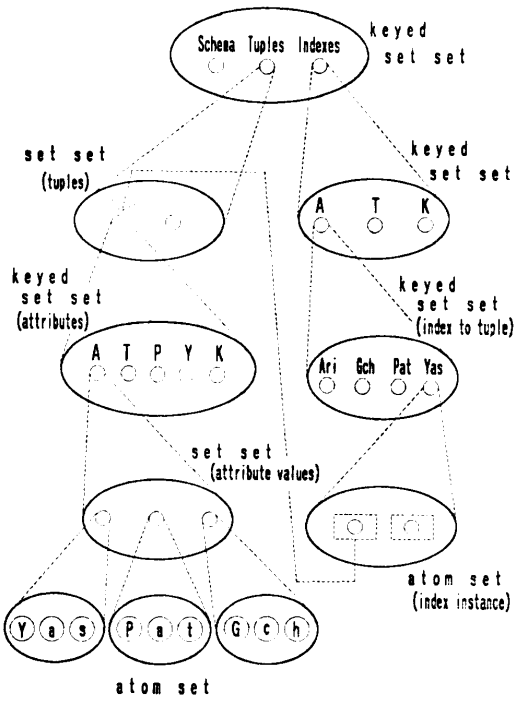

図 4 NFR データベースの概念構造
Fig. 4 Conceptual structure of NFR database.

長は異なってくる．

ここで，データベースの構成要素を AAM の集合に当てはめてみる．この様子を**図 4** に示す．まず，タプルの集合を “tuples” という名前の集合集合で表す．次にそのそれぞれのタプルの下位に “attributes” という名前の集合を定義する．この集合は，キー付き集合集合で，A, T, P, Y, K のキーを持つ五つの要素からなる．これは，各タプル中の属性に相当する．このキーを持つ要素はそれぞれアトム集合であり，アトムに文字としての意味を与えることで，アトム集合を属性値の文字列に対応させる．つまり，タプルの集合は，タプルの内部構造を集合の階層に順次展開したものである．

一方，ある属性の値からインデクスファイルを用いてタプルにアクセスしたいという場合があるが，この例では，下位にアトムとして集合 ID を持つキー付き集合集合を用いている．キー “Yas” を持つ要素は，常に属性値 “Yas” を持つ全部のタプルを指す．このような検索は，例えば次のように表される．

micro (search (T, direct (current-element (
micro (search (“Yas”, micro (search (“A”,
micro (search (“Indices”))))))))).

このデータベースへの他の検索や更新も，AAM の

オペレーションの列で容易に表すことができる.

4. AAM の実現方式

我々は以上に述べてきた AAM の形式モデルに基づき，その実現性を検証するため，マイクロコンピュータ上でこのアクセス法の試作を行った．この章では，AAM の試作システムで用いた実現方式について述べる.

4.1 AAM の試作システムの実現方針

AAM は，基本的には重み付きBトリーという新しい技法で実現される．一般に，Bトリー[8] はインデクスファイルの作成に使われる．しかし，AAM ではこのようにキーによるアクセスだけでなく順序を用いた効率の良いアクセスが必要となるため，サブトリーの要素の数（重み）の情報を持ち，これでトリーをたどれるようにする．これが重み付きBトリーである．すなわち，アトム集合や集合集合ではトリーはキーの代わりに重みの情報を持ち，キー付き集合集合ではトリーはキーの情報と重みの情報の両方を持つ．これによって図2の AAM オペレーションの階層的な体系が実現される．要素の挿入や削除が起きた際に分割や併合の再編成を行うのは，Bトリーの技法と全く同じである.

AAM では，集合の内部も集合間の階層も均一なトリー構造であり，集合の階層全体で大きな一つのトリーをなす．ただ一つの例外はルートであり，ルート自身がルートを含むという構造を実現するため，再帰的なブート部分が存在する.

4.2 AAM の試作システムの構成

AAM の試作システムの実現には，次の3レベルの ADT オペレーション階層とバッファリングモジュールによる構成を用いた(図 5).

（1） AAM プリミティブオペレーション.
（2） トリーオペレーション.
（3） ノードエントリオペレーション.

それぞれのレベルでは，オペレーションは，下位のレベルのオペレーションの組合せとして実現される．ADT により，データタイプはシステムの各々の階層に局所化できる．これにより，ある特定の ADT を変更するときは，システムのある限られた部分だけを見直せばよいことになる．AAM のプリミティブオペレーションについては，第2章で述べたので，残りの二つについて以下に述べる.

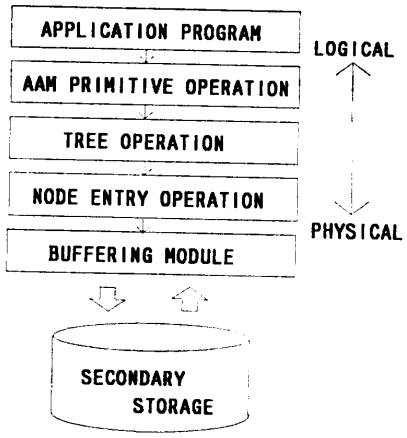

図 5 AAM の試作システムの構成 (ADT 階層)
Fig. 5 ADT hierarchy of AAM prototype system.

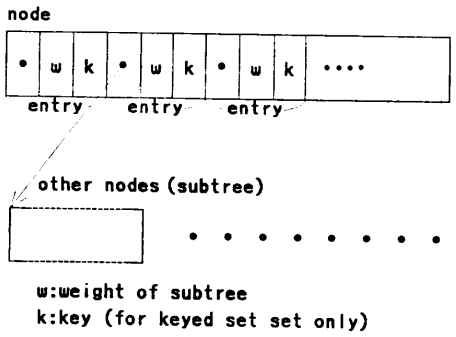

図 6 重み付きBトリーの構造
Fig. 6 The structure of "weighted B-tree" (a new variety of B-tree realizing "order" in a set).

4.2.1 トリーオペレーション

AAM のプリミティブオペレーションは，トリー構造で実現される．ADT としてのトリー構造の一般的な記述はよく知られているので[6]，ここでは省略する．AAM のトリーは，ノードとそのノードに属する複数のエントリの順序列からなる．各エントリはサブトリーをなす子ノードを指し，子ノード以下のサブトリーの要素数（重み）の情報を持つ．キー付き集合集合では，各エントリは，さらにサブトリー内の最大キーの情報も持つ（図 6）．一つの集合を表すトリーは，トリーの頂点のルートノード，末端のリーフノード，および，そのどちらでもない中間ノードからなる．要素数が少なく，1レベルのトリーでは，ルートノードはリーフノードでもある．この場合，中間ノードは存在しない.

トリーのルートノードは集合そのものを表し，その集合に関する情報を持つ．一方，リーフノードは集合の要素の情報を持ち，下位の集合のルートノードを指

す．ただし，アトム集合ではリーフノードのエントリは，アトムそのものである．

4.2.2 ノードエントリオペレーション

前章で述べたトリーオペレーションは，通常，物理的な I/O 単位のブロックを一つのノードに対応させることで実現する．しかし，AAM では，ある集合を構成する要素の数が少ないこともあり，一つの集合は一つのトリーを形成するので，必ずしも効率の良い実現方式とはいえない．そこで ADT によるシステム設計の特長を生かし，AAM ではトリーオペレーションの下位の実現方式を変えてこの問題の解決を行う．これが，ノードエントリオペレーションである．

これは，ノード内のエントリをリストの要素としてそれぞれ別々に物理ブロックに割り付ける方式で，エントリの挿入，削除などはこのリストのオペレーションで表現する．この方式では，エントリごとに領域を確保するため，同一ノードのエントリが多数の物理ブロックにまたがり，I/O 回数が増加する恐れがある．これを避けるため，同一ノードのエントリどうしは，空きがあれば必ず同一の物理ブロック内に配置する．すなわち，ある集合を構成する要素が少ないとき，物理ブロックは複数のノードに共有され，また，多数の要素からなる集合のノードは，ブロック全体をほぼ占有するようになる．

また，アトム集合のリーフでは，エントリは一つ一つのアトムに対応するため，内部的には配列で実現してポインタ領域のオーバヘッドをなくす．したがって，アトム集合のリーフは連続した領域にあり，必ず同一の物理ブロック内に入れられる．しかし，この場合も上位レベルに対する ADT オペレーションは同一である．

4.3 AAM の試作システムの効率評価

以上のような試作システムの構成を通しての AAM の効率の評価についてまとめる．まず実行速度であるが，ここでは，一つのノードへのアクセスを 1 として，要素数 n の集合におけるオペレーションのオーダの評価を行う．物理ブロックへのアクセスの回数もこのオーダに準じると考えられる．

(1) $O(\log n)$ のオペレーション

次の三種類のオペレーションは，ルートノードからリーフノードまでの間をたどったり，各ノードのキーや重みを更新したりするので，実行効率はトリーの高さ，すなわち n の対数のオーダとなる．

更新オペレーション

カレント要素変更オペレーション

カレント集合変更オペレーション

(2) $O(1)$ のオペレーション

次のオペレーションは高々カレント要素を含むリーフノードを参照するだけなので，オーダは 1 となる．

カレント情報オペレーション

以上の実行オーダは，理論的な予測によるものであるが，試作システムによる実測を通しても，ほぼ同様の結果が得られている．

最後に，スペース効率の評価を行う．上述の理論的なオーダが保証されても，実際のアプリケーションの性能はスペース効率によって大きく左右される．ノードエントリオペレーションの横方向のポインタ部のオーバヘッドは CDR コーディング[9]の技法を用いれば実際上無視できるようになる（現プロトタイプでは未採用）．したがってノード内の各エントリ長を既存の B トリーと比較すればよい．これは，AAM オペレーション階層（図 2）の最も一般的な形式であるキー付き集合集合で，

$$\frac{\text{キー部領域}+\text{重み領域}+\text{子ポインタ領域}}{\text{キー部領域}+\text{子ポインタ領域}}$$

となり，重み領域の部分だけ従来の B トリーより効率が落ちるようになる．しかし，ノードエントリオペレーションにより，複数トリーのノードが同一ブロックに入るため，逆にコンパクション効果もある．今後，アプリケーションとの対応を含め，シミュレーション等を通して効率評価を詳細化していきたい．

5. おわりに

本論文では新しいファイルシステムとそのオペレーションのフォーマルなモデルを提案し，また，その実現方式についても述べた．著者らは数種のミニコンピュータ，パーソナルコンピュータ上で AAM の試作システムを作成した．

現在，このシステムの評価を行っているが，AAM は本文で述べたように単純な構成で柔軟な機能を実現できるため，ユーザはデータ処理プログラムを容易に短期間で開発できることが分かった．このことを利用して，様々なデータタイプやアドホックな検索を扱える，コンパクトなデータベース管理システムの開発を現在計画中である．

また，本論文では最も基本的なファイルオペレーションに対して抽象化を行い，体系化することができたが，さらに上位概念としてデータベースのモデルを考

えた場合のオペレーションも含めて整合性のある全体の体系を確立していくよう研究を進めている．

参考文献

1) Lorie, R. A. and Nilsson, J. F.: An Access Specification Language for a Relational Data Base System, *IBM J. Res. Dev.*, Vol. 23, No. 3, pp. 286-298 (1979).
2) Stonebraker, M. et al.: The Design and Implementation of INGRES, *ACM Trans. Database Syst.*, Vol. 1, No. 3, pp. 189-222 (1976).
3) Aho, A. V. et al.: *Data Structure and Algorithms*, Addison-Wesley, Mass. (1983).
4) IBM: Functional Structure of IBM Virtual Operating Systems, *IBM Syst. J.*, Vol. 12, No. 4, pp. 368-423 (1973).
5) Bell Telephone Lab.: *UNIX™ Time-Sharing System, Unix Programmer's Manual, 7th ed.*, Bell Telephone Lab. (1979).
6) Aho, A. V. et al.: *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Mass. (1974).
7) Arisawa, H. et al.: Operations and the Properties on Non-First-Normal-Form-Relational Databases, *Proc. VLDB*, pp. 197-204 (1983).
8) Bayer, R. and McCreight, E. M.: Organization and Maintenance of Large Ordered Indexes, *Acta Informatica*, Vol. 1, No. 3, pp. 173-189 (1972).
9) Clark, D. W. and Green, C. C.: An Empirical Study of List Structure in LISP, *Comm. ACM*, Vol. 20, No. 2, pp. 78-87 (1977).

(昭和61年5月15日受付)
(昭和62年5月13日採録)

有澤　博（正会員）

昭和23年生．昭和48年東京大学理学部物理学科卒業．富士通(株)を経て昭和50年横浜国立大学工学部に奉職．現在同学部電子情報工学科助教授．工学博士．データベース理論，データベース・システム・アーキテクチャを研究テーマとしている．著書として「データベース理論」(情報処理学会) 等．情報処理学会誌編集委員．電子通信学会会員．

岡山　利次

昭和32年生．昭和55年東京大学農学部農業生物学科卒業．横浜国立大学環境科学研究センター研究生を経て，昭和56年日立ソフトウェアエンジニアリング(株)入社．以来，次世代データベースシステムの研究に従事．現在，バイオテクノロジー関連のソフトウェアの研究，開発を担当．情報科学とかつての専攻の生態学との境界領域の開拓に意欲を持つ．

鈴鹿　豊明

昭和35年生．昭和59年横浜国立大学工学部電気工学科卒業．同年，日立ソフトウェアエンジニアリング(株)入社．在学中よりデータベースシステム，データモデルの研究に従事．

久保　隆（正会員）

昭和18年生．昭和42年立命館大学理工学部数物学科卒業．同年日立電子エンジニアリング(株)入社．現在日立ソフトウェアエンジニアリング(株)主任研究員．人工知能，自然言語理解の研究に従事．ACM 会員．