DB 29 － 2
(1976. 7. 8)

抽象データ型とデータ独立性
―ACM Conference on Data―

千葉恭弘
(日本ユニバック総合研究所)

0. はじめに

1976年3月22日より24日まで米国ユタ州ソルトレイク市において，ACMのConference on Dataが開催された。この会議はACMの2つの特別グループSIGPLANとSIGMODの共催で，データに関する"抽象・定義及び構造"が主テーマであった。会議の意図は，従来独立に論じられてきたプログラミング言語におけるデータの概念とデータベースの分野におけるデータの概念との間に共通点を模索し，互いに交流を図ることであった。この意図を反映し，会議の参加者も言語とデータベースの関係者が半々を占め，発表論文もデータの抽象化に関するものが大部分であった。この会議に出席する機会を得たので，本稿では会議での中心的な話題であった抽象データ型とデータ独立性の問題に関して発表論文の紹介と研究動向の報告を行う。

1. 抽象データ型

最近のプログラミング言語の分野では，ソフトウェア工学の課題である信頼性のあるソフト開発(reliable software)に関係して，プログラム作成技術の研究が盛んである。Dijkstra[1]の"構造的プログラミング"を始めとし多くの提案が，信頼性のあるソフト開発のためにはプログラムを人間が管理しやすいようなより良く構造化された形式に構成していく必要性を強調している。このようなプログラム作成の手段として，プログラムの論理的な機能の抽象化から始めて順次その具体化に進むいわゆる下降的(top down)アプローチや逐次詳細化(stepwise refinement)のアプローチが提唱されている。抽象化(abstraction)から具体化(representation)への段階的接近の概念は，Parnas[3]の主張する"情報隠し"(information hiding)の思想に通じる。すなわちシステム設計に関して従来は野放しであった各種の情報を必要な所にだけ供給し，システム内のモジュール間のやりとりを簡潔にする。各モジュールのインプリメントに関する情報はそのモジュールの中に閉じ込めてしまい外部からは参照できないようにする。

このようなアプローチはプログラムの制御構造(control structure)に関する抽象化に適用され，多くの興味ある論文が発表されている。制御構造の抽象化が進む過程でプログラムで処理するデータに関しても同じような抽象化の必要性が認識されてきている。実際，構造的プログラミングや高次言語(very high level language)の支援のためにはデータの抽象化が必須のものと考えられている。

ここ数年，データの抽象化の手段として抽象データ型(abstract data type)の議論が活発になされている。抽象データ型の定義はParnas[4]に従うと次の5つのアプローチに分類される。

① 文法的定義(syntactic)：データ型は宣言文の中で変数に関して与える情報である。例えば VARIABLE X IS INTEGER.

② 値空間(value space)：データ型は潜在的にとり得る値の集合によって定義され，数学的集合操作の対象となる (Hoare[2], Wegbreit：EL1)。

③ 行動(behavior)：データ型は値空間とその空間上の要素に作用する操作(operation)の集まりで定義される (Naur：ALGOL 60)。

④表現（representation）：データ型はより基本的（primitive）な型を用いて表現され，最終的にはハードウェアもしくはコンパイラがインプリメントしている基本型に帰着させる（Van Wijngaaden：ALGOL 68, Wirth：PASCAL）。
⑤表現プラス行動：データ型は表現と，その行動を定義する操作の集合で定義される（Dahl：SIMULA 67, Liskov：CLU）。

最近のプログラミング言語の研究動向では，抽象データ型を⑤の表現プラス行動として定義する方向が主流となっている感がする。この方式では，1つのデータ型を定義する個有の表現とその表現上に操作する行動の記述が独立の単位を構成し，カプセル化される。このカプセルの概念は制御構造の抽象化における手続き（procedure）と対応し，次のような利点を持つ（Horning [10]）。

①繰り返し（repetition）の回避：同種な行動をとるデータのグループ間でコードの共用（code sharing）が可能であり，類似な宣言・定義を省略できる。
②モジュラ・プログラム構造：制御構造の抽象化はアルゴリズムの"抽象操作（abstract operation）"への分割を行うが，これに対応し複雑なデータ構造もカプセルによって具現化される"抽象対象（abstract object）"へ分解させることにより，モジュール化の利点を実現できる。
③構造的プログラミングの基盤：データ構造の逐次具現化と具現化情報の埋め込めにより，構造的プログラミングを支援する。
④概念の単位：カプセルが何を表現しているを明確にし，如何に表現されているかを遮蔽することにより，プログラムの理解と理由づけに関する概念的な単位を構成する。
⑤インターフェイスの詳細記述：1つのカプセル内での対象に関する操作を明確に記述する（interface specification）。
⑥保守・改良の単位：1つのカプセル外の変動要因には原則として依存しない。
⑦言語拡張機能：プログラマが自由に定義できる新しいデータ型の機能により言語の拡張が可能である。
⑧独立コンパイルの単位

このようなカプセル化の思想を持つ抽象データ型は，現時点でSIMULA 67のclassの概念，CLUのclusterの概念，ALPHARDのformの概念などとして実際にインプリメントされている。

抽象データ型に関する種々の問題点はFlonのサーベイ論文[5]によると次のような点が指適されている。

1)新しいデータ型の定義（type extension）
2)操作のカプセル化（encapsulation）
3)名前の外部伝達（exportation）
4)型と構造（type vs. structure）
5)型定義のパラメータ化（parameterization of type definition）
6)ポインタ
7)型の自動変換（coercion）
8)リテラルの扱い
9)初期設定（initialization）
10)代入（assignment）

今回のConference on Dataでの抽象データ型に関する論文でも，これらの点に関係する議論が多い。Parnas et al.[4]ではデータ型拡張の目的として，抽象

化の推進，コンパイル時点での型検査の充実，プログラムの簡略化（コード共用化），データの携帯性（portability）の向上などを掲げ，これらの目的の達成のためには，前述した抽象データ型定義の5つのアプローチの他に変数のクラスとして定義する新しいアプローチを提唱する。これはコード共用化の観点でみると，従来のデータ型の定義では変数が特定のデータ型に拘束され過ぎ，異ったデータ型でも類似の行動を示すものに関して相互の代替を認めることができず，コード共用化が阻害されているとするもので，これに代ってより制限の少ない定義の必要性を指適している。彼等のポロポーザルでは，変数（variable）をprimitiveとし，データの表現と行動が同一であるすべての変数を同一のモードと呼ぶ。データ型はこれらのモードのクラスで定義される。必要とされるデータ型の分類には次の4つが掲げられている。

a) spec-type ：外部に見える行動が同じであるようなモードから成る。すなわち特定のinterface specificationを充すデータ型である。
b) rep-type ：表現が同じであるモードから構成される。
c) param-type ：パラメータ化されたモード記述を呼び出すようなモードから成る。
d) variant-type：共通の属性を持つモードから成る。

以上の議論は，従来の型定義がモードのレベルであり，厳格なコンパイル時点の型検査を緩和し弾力的なコード共用化を推進するためにもう1つ上のレベルをデータ型として定義したものであると考えられよう。

既存のプログラミング言語の上でこれらの抽象データ型を実現しようとする試みが発表されている。GriesとGehani［11］はプログラミング言語PASCALの中に行動記述を伴ったユーザ定義によるデータ型の思想を導入し，操作や手続きの"積み上げ（overloading）"の概念を提唱している。WellsとCornwell［12］はプログラミング言語Madcap 6に導入したデータ型カプセル化のスキーマを論じベースの言語における operator（ユーザ定義のSPACE上で作用する）と手続き呼び出しの概念的区分が表現力の改善に役立つことを述べている。

"情報隠し"及び名前の外部伝達に関してKoster［13］はモジュール間の"可視性（visibility）"の問題をとりあげている。プログラムをユニットに分割し，何らかの対象に関して"定義（define）"ユニットと"適用（apply）"ユニットを区別する。この概念に基づきユニット間の可視性規則を定義し，可視性によってユニット群を構造化する方法を提示している。

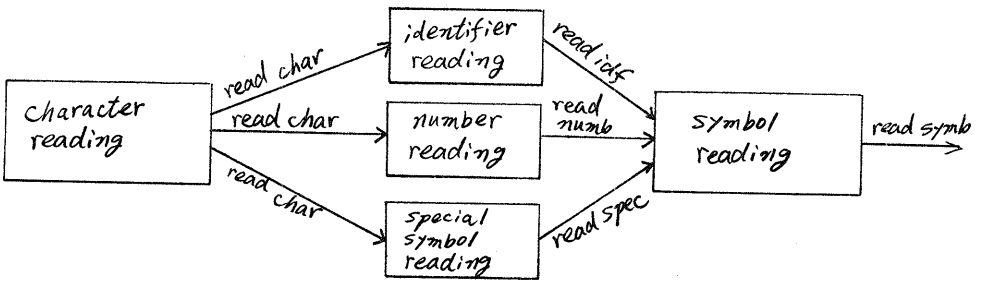

図1　ユニット間の可視性

2. データ独立性

データ対象の抽象化をその具体化から区別するというプログラミング言語における抽象データ型の議論は，データベースの分野におけるデータ独立性の概念に相当する。データ独立性の概念は，データベースを利用するアプリケーション・プログラムの側ではデータベースの物理的な構成を知ることなしに純粋に論理的局面だけを考慮してデータベースとインターフェイスをとることを可能とする。多目的共用化ファイルとしてのデータベースではこのデータ独立性の概念は必須の機能と考えられているが，実際にはこの概念が多様な側面を含むため完全なデータ独立性の達成はかなり難しいものと思われる。現状ではデータベースの定義を幾つかのレベルに分け，各レベルのインターフェイスをデータベース管理システムの制御の下に置くことによりデータ独立性を実現しようとしている。図2はANSI/SPARCによるデータベースの定義情報の3つのschemaの関連図を示している。Conceptual schemaはデータベースの全体的枠組を定義するもので，基本的なデータ模型を表現している。External schemaは1つのデータベースを囲む各々のアプリケーション・プログラムのデータ視点（user's view of data）を表現するもので，conceptual schemaでのデータ模型に対する部分模型（sub model）を構成する。Internal schemaはデータがデータベース内部でどのように格納されているかを定義するもので，表現に関する情報を与える。

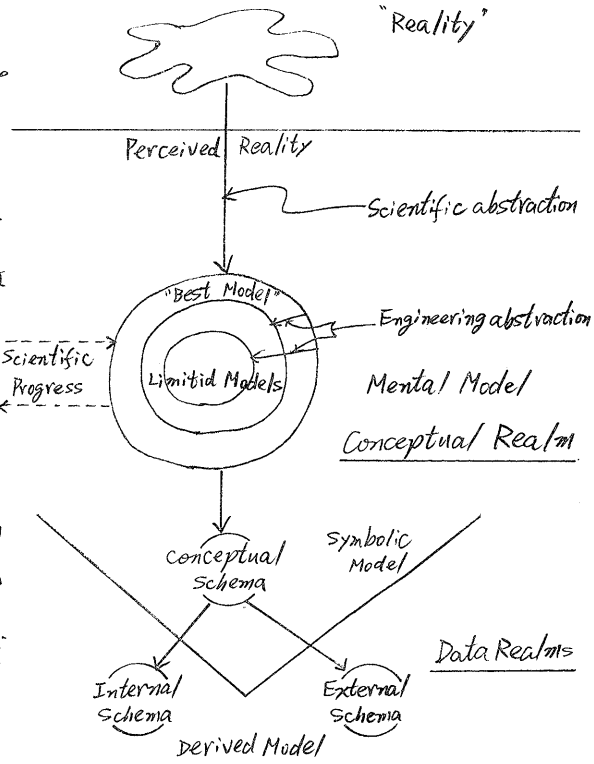

図2　ANSI/SPARCによる3つのschema

データ独立性はデータベースの構成要素に何らかの変更が与えられた時にアプリケーション・プログラムにどのような影響が生じるかによって判定される。アプリケーション・プログラムはexternal schemaに結びついているため，良好なデータ独立性の達成のためにはconceptual schemaおよびinternal schemaの変更がexternal schemaに与える影響を最小限に抑えることになる。

3. 抽象データ型とデータ独立性

Hammer, [15] は抽象データ型とデータ独立性の間に多くの類似点があることを指摘し，データベースのデータ独立性は行動定義を含む抽象データ型のアプローチを導入することによりある程度推進させることができると述べている。しかし抽象データ型とschema記述の間には次のような格差がある。すなわち概念的なデータ構造を定義する現在のデータ模型のアプローチでは，データ型の表現を記述しているのではなく，あるレベルの構造を記述しているのであり，一方抽象データ型では表現記述はあるが，どのレベルでも構造の概念をユーザに与えるわけ

ではなく，データ型の意味(semantics)は完全に行動によって定義される。従って抽象データ型の概念が形成されたプログラミングの環境では，抽象から具体化への規則によりモジュール別けが良くでき，データ対象の行動的意味の把握から出発することができる。操作はそのデータ対象を特性化することができ，その具体化は完全にカプセル化することができる。これに対しデータベースの環境では実用されるデータベースの意味は，含まれているデータそのものによって規定されるのであって，データベースに適用される特定の操作の集合で定義することは困難である。このような観点からHammerは抽象データ型のアプローチを適用できる分野としてデータベースの整合性判定(integrity check)を掲げている。データベースのconceptual schemaはデータベースの最も一般的な意味を表現しているので，このレベルでの行動仕様(operational specification)はデータ上の意味のある操作のクラスを制限したり，特性化したりするだけであり，external schemaで定義される操作は従って conceptual schemaで供給される一般的な形式(template)を個別化(instantiation)するものとなる。

Hammerの観点はMcLeod [16]の関係形式模型における領域定義言語(domain definition language)に反映されている。関係形式模型(relational data model)における意味の整合性(semantic integrity)には次の3つのレベルが設定される。

① 領域定義：atomic data valueの抽象的な集合に関する記述

② 関係構造記述(relation structure specification)：base relationの基本的構造の記述で，その関係を構成する属性の名前付けと，その属性の定義域などが記述される。

③ 関係の制約条件(relation constraint)：関係の基本キー(primary key)などの付加的な性質が記述され，関数的依存性(functional dependency)などを支援する。

今回の発表では①の領域定義言語の構文が提出された。この領域定義言語は，

a) 領域を構成するatomic data valueの集合の記述

b) 領域定義が充たすべき条件の記述 (Description & Ordering)

c) 領域定義が充たされない時の行動記述 (Violation-action)

の3つの部分からなり，抽象データ型のアプローチを意味の整合性判定に適用したものと考えられる。

```
domain NAME                                          ("Smith, John")
     description
          last: string
          ', '
          first: string
     ordering
          last, first
     violation-action
          error

domain SEX                                           ("female")
     description
          oneof 'female', 'male'
     ordering
          none
     violation-action
          error 'sex must be female or male'

domain MONEY                                         ("$100")
     description
          '$'
          value: number where >=0
          where length(right(*, '.'+1))=2
                or not present *, '.'
     ordering
          value
     violation-action
          substitute null 'value in error, null has been assumed'
```

```
domain ITEM                                          ("AB-75-326")
     description
          string where not has numerics, '-'
          i1: '-'
          i2: string where not has alphabetics, '-'
          where repititions i1 through i2 >=1 and <=3
     or
          string where call check_item
     ordering
          call compare_item
     violation-action
          substitute left(*, 5)

domain DATE                                          ("1/28/76")
     description
          month: number where >=1 and <=12
          '/'
          day: number where integer and >=1 and <=31
          '/197'
          year: number where integer and >=5 and <=9
          where (if (month=4 or =5 or =9 or =11) then day<=30)
                and (if month=2 then day<=29)
                and (if (month=2 and year^=6) then day<=28)
     ordering
          year, month, day
     violation-action
          error
```

図3 領域定義言語の例

4. データベースの抽象化

プログラムの制御構造の抽象化に用いられている下降的アプローチや逐次詳細化アプローチの手法をデータベースの抽象化に適用しようとする動きがある。ここでデータベースの抽象化とは，データベースに格納されるデータ要素間の関係（データ構造）を抽象レベルから具体化する方法を意味し，実際にはconceptual schemaの設計を下降的な手法で段階的に進めることになる。Senko [17] は従来提唱してきたDIAM IIモデルを下降的なシステム構成法として紹介している。今回の発表ではシステムの構成をANSI/SPARCの3つのschemaに対応したEND-USER level, INFOLOGICAL level, DATALOGICAL levelの3つレベルで構成し，INFOLOGICAL levelでのデータ要素間の構造を連想対（association pair）の概念で記述し，そのデータ操作言語FORALを解説している。

Smith [18] によるデータベースの抽象化は，データベース設計者がユーザの要求を突つめて最も抽象的と考えられるデータ対象は何かを把握するところから出発する。この抽象的なデータ対象はそれ自身何らかの構造を含んでいる可能性がある。設計のステップはこれらの抽象的データ対象を逐次より単純なデータ対象間の関係に分解していくことになる。分解が完了した段階では最初の抽象的データ対象は幾つかの関係上の階層構造となり，各レベルの関係はCoddの第3正規形（Third Normal Form）を構成することになる。Smithはデータベースの構造化のprimitiveとして"関係（relation）"というデータ型をPASCALに導入し例示している。関係データ型は次のように定義され，図4でデータ構造が示される。

```
type tuple = record
               d1: d1 type;
               d2: d2 type;
                 ⋮
               dn: dn type
             end
type relation = set of tuple;
```

```
type T = relation key [Si1, ···, Sim]
           S1: {key of} T1;
                 ⋮
           Sn: {key of} Tn
         end
```

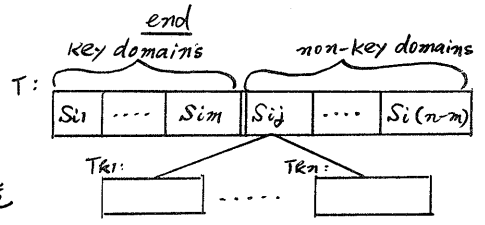

図4 関係定義とデータ構造

Bubenko et.al. [19] はschema設計の下降的アプローチを同じように提唱しているが，彼等の方法はユーザの情報要求から情報構造を導き出すという逆の方向をとる。例としてDBTG 機型のschemaを設計する方法が示されている。

Schema 設計のプロセスは次の3つのタスクを含んでいる。

① 情報構造と関係の決定
② schema候補の設計
③ schema候補の分析と評価

①のタスクでは現実に関する数量的な情報を収集し，現実の状況を情報構造にマップする。情報構造のprimitiveは対象（object）で，同種の対象は対象クラスを構成する。対象クラスの全ての集合 B=（C1, ···, Cn）の部分集合 Biは特定のアプリケーションで意味を持つものと考える。2つの部分集合BiとBjの間には集合の包含関係により順序が定義される。これを因子（constituent）関係

と呼び（＜）で表示する。

$$B_i\ (<)\ B_j \iff B_i \subset B_j$$

因子関係には推移律が成立する。B_i に関して因子関係が成立するものの集合を $<(B_i)$ として表示する。$<(B_i)=\varnothing$ であるならば B_i は *initial* と呼ばれる。1つの対象クラス B_i に対する検索は B_i^1 と B_i^2 に関するものと考えることができる。ここで $B_i = B_i^1 \cup B_i^2$，$B_i^1 \cap B_i^2 = \varnothing$ であり B_i^1 は unique identifier を持つもの，B_i^2 は B_i^1 からの関係を辿って検索されるものである。このような基本的な情報構造と関係を用いて②のタスクでは幾つかの schema 候補が設計される。Schema 設計の第1の候補はデータベースに格納される対象が *initial* だけにするもので，他はすべてこれから実行時に抽出する。もしくはすべての B_i をデータベースに格納する。この2つは極端な schema であり，現実には response time 指向的な設計や更新指向的な設計があり。それぞれの要請を満すような設計ステップが提唱される。

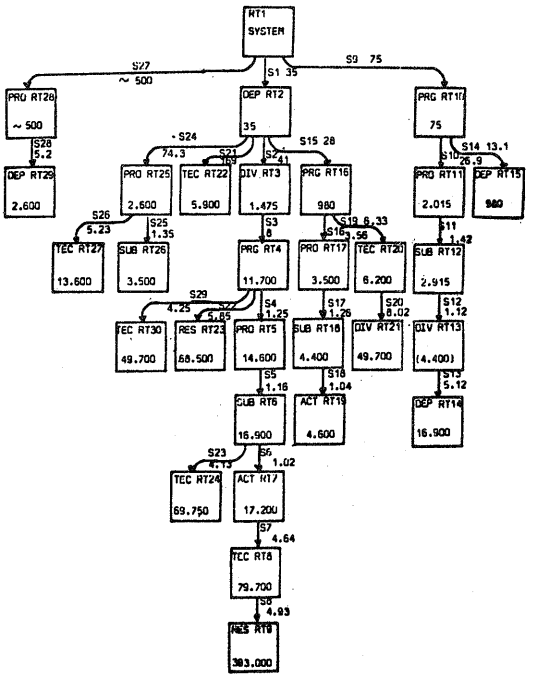

図5. response time 指向 schema 設計例

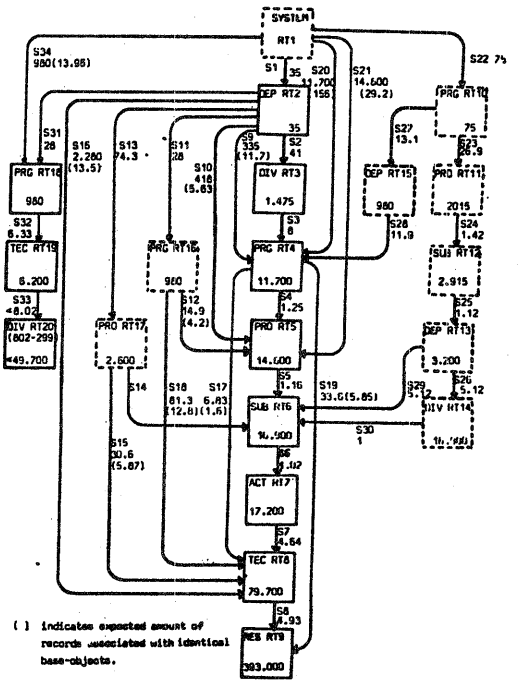

図6. 更新指向 schema 設計例

5. おわりに

今回の会議では，抽象データ型とデータ独立性の議論が共通点を認識しながらも良くかみ合って相互交流をしたという感じからは遠いものであった。データベースの分野で抽象データ型のアプローチを適用しようとした場合には Hammer の指摘のように非常に難しい問題点が存在する。これは主にデータベースが共用を前提としたデータの蓄積場であることから発生するものであろう。しかしデータ独立性の概念はまさにデータベースの共用化のための必須の手段として考えられているのであり，抽象データ型のアプローチをデータ独立性の達成手段として適

用可能であるか否かの議論がなされてしかるべきであろう。

従来から主張されているsubschemaの概念に抽象データ型の手法が適用できるのではないかと考える。1つの試案として次のような構成を考えてみた。データ記述（定義）部分を細分化して5つのモジュールとして相互に独立性を与える。

① ベース・データ定義モジュール
従来のschemaに相当する部分で，データベースに格納されるデータを定義する。ここで定義されたデータをベース・データと呼ぶ。

② 記憶構造定義モジュール
記憶構造を定義しベース・データとの対応を与える。

③ 論理スペース定義モジュール
アプリケーションの論理的なデータ視点を定義し，これを論理スペースと呼ぶ。同時にこの論理スペース上で定義される操作（operation）の名前をリストする。この論理スペースはここで指定された操作以外では触ることができないようにする。

④ ベース・スペース・マップ・モジュール
論理スペース上の要素とベース・データとの対応を定義する。これは1対1の対応以外に何らかの加工によって生成される仮想的なものでもよい。またベース・データの対応をとる際の条件を記述できるものとする。

⑤ 操作定義モジュール
論理スペースで指示された操作を手続き（procedure）として記述する。また外部伝達の認められる操作名を定義し，それ以外はこのモジュール内での一時的関数となる。

一般のアプリケーション・プログラムでは対象とする論理スペースを指示し，このスペース上だけで処理を考える。各モジュールの変更ができるだけ他のモジュールに伝わらないようなカプセル化を留意するが，影響が波及する場合は対応を定義するモジュールで吸収するようにする。④の方法としてDate & Hopewell [20] が提唱したような論理ファイル定義言語が考えられる。

参考文献

[1] Dijkstra, E.W., 'Notes on Structured programming', Structured Programming, by O.J.Dahl, E.W.Dijkstra and C.A.R.Hoare, Academic Press, 1972

[2] Hoare, C.A.R., 'Notes on Data Structuring', Structured Programming, Academic Press.

[3] Parnas, D.L., 'Information distribution aspects of design methodology', Proc. IFIP 1971, 339-344

[4] Parnas, D.L., Shore, J.E., Weiss, D., 'Abstract Types Defined as Classes of Variables', Proc. Conference on Data, SIGPLAN Notices Vol.11, 1976

[5] Flon, L., 'A Survey of Some Issues Concerning Abstract Data Types', CMU-CS, 1974

[6] Liskov, B., Zilles, S., 'Programming with Abstract Data Types', SIGPLAN Notices, Vol.9, pp 50-59, 1974

[7] Ichbiah, J.D., Morse, S.P., 'General Concepts of the SIMULA 67 Programming Language', Ann. Rev. of Automatic Programming 7, 1972

[8] Wirth, N., 'The Programming Language PASCAL', Acta Informatica 1, 1971
[9] Wulf, W.A., 'ALPHARD: Toward a Language to Support Structured Programs', CMU-CS report, 1974
[10] Horning, J.J., 'Some Desirable Properties of Data Abstraction Facilities', Proc. Conference on Data, SIGPLAN Notices Vol. 11, 1976
[11] Gries, D., Gehani., 'Some Ideas on Data Types in High Level Languages', Supplement to the Proceedings of Conference on Data, 1976
[12] Wells, M., Cornwell, F., 'A Data Type Encapsulation Scheme Utilizing Base Language Operators', Proc. Conference on Data, SIGPLAN Notices Vol. 11, 1976
[13] koster, C. H. A., 'Visibility and Types', Proc. Conference on Data, 1976
[14] Steel, T.B, 'Data Base Standardization: A Status Report of ANSI/SPARC', Data Base Description, by Nijssen, D., Proc. IFIP TC-2, 1975
[15] Hammer, M., 'Data Abstractions for Data Bases', Proc. Conference on Data, 1976
[16] McLeod, D. J., 'High Level Domain Definition in a Relational Data Base System', Proc. Conference on Data, 1976
[17] Senko, M.E., 'DIAM II: The Binary Infological Level and its Database Language - FORAL', Proc. Conference on Data, 1976
[18] Smith, J. M., Smith, D. C. P., 'Data Base Abstraction', Supplement to the Proceedings of Conference on Data, 1976
[19] Bubenko Jr, J.A., Berild, S., Lindencrona-Ohlin, E., Nachmens, S., 'From Information Requirements to DBTG-Data Structures', Proc. Conference on Data, 1976
[20] Date, C. J., Hopewell, P., 'File Definition and Logical Data Independence', Proc. 1971 ACM-SIGFIDET Workshop, 1971
[21] 鳥居, 杉藤, 真野, 二木, 'プログラム作成技術の現状に関する調査報告 [I]', 電子技術総合研究所調査報告 第185号, 1975