计算机科学1998Vol. 25№. 6

高级程序设计语言中归并数据类型的研究

A Study of Union Data Type in High Level Programming Languages

王汝传

（南京邮电学院计算机科学与技术系　南京210003）

摘　要　本文讨论如何用归并类型来代替高级程序设计语言中指针和记录（如 PASCAL）。归并类型的设计除了具有统一之外，还具有更高可靠性，本文已将归并类型插入到 Modula-2程序语言中。

关键词　高级程序设计语言，数据类型，归并数据类型，指针，记录

1　引言

在许多高级程序设计语言中都使用指针数据类型，但指针和转向语句一样，是面向计算机设计的。转向语句在第一批面向问题的程序设计语言（如 FORTRAN、ALGOL 语言）中出现后，其局限愈来愈多，以致后来一些程序设计语言中如 PASCAL、C 等，出现指针数据类型。指针使得程序设计人员在一个低的抽象级上进行操作，在链接结构运行、变化的同时，还必须兼顾显式请求和存储空间的释放。使用指针往往还会遇到以下问题：

（1）对于一般变量值，人们立即会看出这些值有无意义，而对于指针变量，人们很难知道它们是否具有所需的值。

（2）对于出错指针的存取一般是难以识别的。

（3）如果操作系统不提供合适的保护机构，对于无意义的值间接引用指针进行赋值，就可能使程序代码出现切口。

自从 PASCAL 语言出现以后，人们就开始检查程序设计语言中的数据类型有无其数学基础，我们可以用递归数据类型，来代替指针类型，例如，元素表类型定义为：一个表可以是空的，或者由一个 Element 类型的元系和一个表构成：

Type List＝VOID|Element X List

其中"X"是用于类型笛卡尔乘积。若用 Modula-2语言表示这个数据类型就得使用指针：

```
TYPE List=POINTER TO Record;
     Record=RECORD head;Element;
     tail:List END
```

对于一个二叉制树带有类型 Leaf（叶）的元素由两类节点构成，即由叶和内部节点构成，在内部结点上又可以挂两棵树：

Type Tree＝VOID|Leaf|Tree X Tree

若用 Modula-2语言表示，除了指针还得使用联合（记录）：

```
TYPE Tree=POINTER TO Record;
   Record=RECORD
      CASE node:BOOLEAN OF
      FALSE:terminal:Leaf
      |TRUE:left,right:Tree
      END
      END
```

2　归并数据类型

2.1　数据类型

对于归并数据类型应具有下列目标：（1）递归数据类型可用公式表示和执行。（2）避免在指针设计和联合变体范围内的错误。（3）执行一个自动的存储器分配和半自动的存储器清除（无用单元收集）。

对于归并类型可给出下列简式：

归并类型＝指针＋联合变体＋可靠性

归并类型和借此而定义的操作标记用的语言手段依赖于 Modula-2语法，一方面是因为其简明扼要且易懂的表达方式，另一方面是因为归并类型设计置于 Modula-2中执行。

我们可借助归并类型将一个具有二者择一清楚显示未定义二叉树模型化，将类型 Tree 和 List 定义如下：

```
TYPE Tree=|Leaf|Pair
     Pair=RECORD left,right:Tree END;
TYPE List=|RECORD;
     Record=RECORD head:Element;
     tail:List END
```

这里定义 Tree 类型时作为基础类型，同时引入并使用类型 Pair，若在结构上类型相等，则 Tree 类

型可以确定如下：

```
TYPE Tree=|Node|RECORD left,
    right:Tree END;
```

2.2 归并类型使用时的存取

我们考察下列变量的说明：

```
VAR s,t:tree;
    l:Leaf;
    p,Pair;
```

(1)改变显示。若确定一个归并类型变量的类型显示，且变量得到一个值，则类似于使用联合时选定变量所需类型显示。在 Modula-2实现中，首先请求存储空间，相应占用标记字段并在另一语句中直接存取到所需的变体上，例如：

```
t:leaf=l;
t:Pair=p;
```

(2)值的改变，存取一个归并类型变量的值当然是一个复杂的过程，使用归并类型变量时只能在一个保证赋值兼容性的控制结构之内进行存取。存取用的 TYPE 语句，如同一个分情形语句(CASE 语句)，在其标题部有一个归并类型打印输出。此外，在标题部还标明一个符号名，它代表打印输出的值。如果 TYPE 语句之内由于赋值而改变打印输出的显示，则该变量保持不变，如果变量值改变，则变量的对象值也改变。例如，树的遍历：

```
PROCEDURE TraversalTree(tree:Tree);
BEGIN
  TYPE tree WITH node OF
  Leaf:Process(node)
  |Pair:TraversalTree(node.left)
       TraversalTree(node.right)
  ELSE
  END
  END TraversalTree
```

又例如，检索一个列表元素：

```
PROCEDURE  Search ( m:  Element;  list:  LIst ):
                BOOLEAN;
BEGIN
  TYPE list WITH Curnode OF
    Record:RETURN (m=Curnode.head)
      OR Search(m,CurNode.tail)
    ELSE DETURN FALSE
    END
  END Search
```

(3)使用归并类型时进行赋值。如果在变量和一个公用归并类型的输出之间进行赋值，此后变量标识符便有一个涉及赋值源的对象关系，该对象在使用这种赋值方式时不拷贝，对象变量是该对象用的一个别名，这种别名效应特别有益，而且对于实际应用一种程序设计语言必不可少，因为它们使程序的效率极大提高。例如：

S:=t(只要直到 t 或 s 的显示改变，s 和 t 就各有一个涉及同样对象的关系)

t:=NIL(t 如在初始状态下一样未定义，常数 NIL 仍属于虚构的零类型，它是多形的，即可以与所有归并类型分配兼容)。

3 归并类型的语义形式描述

归并类型的语义形式描述通过一个类属的抽象数据类型 Union 说明来进行，该说明不是采集的一个归并类型的一个目标之语义，而是采集该归并类型的所有目标的语义。

$t_1, t_2, \cdots, t_n$ 表示类型显示($n>0$)，而 t_0 表示一个虚构零类型，假设种类 Type(类型标识符)，Value(对象值)，bool 和 dessignator(对象标识符)已知。

3.1 符号

```
init:                                          →union
Create:union x designator                      →union
set-type:union x designator x type             →union
set-value:union x designator x value           →union
share:union x designator x designator          →union
get-type:union x designator                    →type
get-Value:union x designator                   →value
are-shared:union x designator x designator     →bool
```

3.2 语义的构造

借助于类型生成程序设计说明可以简单表示语言的语义。

(1)说明一个归并类型应符合于 init 生成过程，其类型参数 t_1 至 t_n 用选择同一类型占用。

(2)在同样的归并类型的二个对象之间赋值应符合 share 函数。share 函数的第二个参数属于赋值的左面，而第三个参数属于右面。

(3)为了改变一个归并类型变量的值，由于 get-type 函数使控制结构可以模拟，此函数可回送一个对象实际显示。

(4)确定实际值用的函数 get-value 使得一个对象的值变化对那些与该对象有 are-shared 关系的值变化产生影响。

(5)通过 NIL，借助于赋值将一个对象复位到初始状态。

下面通过一个小例子说明具体语言构造上述语义描述，例子中由 union 产生的分项用 q_i 表示。

(1)are－shared 规程

$$\text{are}-\text{shared}(\text{set}-\text{value}(u,d,v),d_1,d_2)$$
$$=\text{are}-\text{shared}(u,d_1,d_2)$$

$$\text{are}-\text{shared}(\text{share}(u,d,e),d1,d2)$$
$$=\begin{cases}\text{true} & \text{if } d=d_1 \wedge e=d_2 \vee e=d_1 \wedge d=d_2 \\ & \vee d=d_1 \wedge \text{are-shared}(u,d_2,e) \\ \text{are-shared}(u,d1,d2) & \text{else}\end{cases}$$

$$\text{are-shared}(\text{set}-\text{type}(u,d,t),d1,d2)$$

$$=\begin{cases}\text{false} & \text{if } d=d_1 \vee d=d_2\\ \text{are-shared}(u,d_2,d_2) & \text{else}\end{cases}$$

are−share(create(u,d)d1,d2)

$$=\begin{cases}\text{false} & \text{if } d=d_1 \vee d=d_2\\ \text{are}-\text{share}(u,d_1,d_2) & \text{else}\end{cases}$$

(2)get−Value 规程

get−value(create(u,d),e)

$$=\begin{cases}\text{get}-\text{value}(u,e) & \text{if } d\neq e\\ \text{error} & \text{else}\end{cases}$$

get−value(set−type(u,d,t),e)

$$=\begin{cases}\text{get}-\text{value}(u,e) & \text{if } d\neq e\\ \text{error} & \text{else}\end{cases}$$

get−value(set−value(u,d,v),e)

$$=\begin{cases}v & \text{if are}-\text{shared}(u,d,e) \vee d=e\\ \text{get}-\text{value}(u,e) & \text{else}\end{cases}$$

get-value(share(u,d1,d2),d)

$$=\begin{cases}\text{get}-\text{value}(u,d_2) & \text{if } d_1=d\\ \text{get}-\text{value}(u,d) & \text{else}\end{cases}$$

(3)get−type 规程

get−type(set−value(u,d,v),e)=get−type(u,e)

get−type(create(u,d),e)

$$=\begin{cases}t_e & \text{if } d=e\\ \text{get}-\text{type}(u,e) & \text{else}\end{cases}$$

get−type(set−type(u,d,t),e)

$$=\begin{cases}t_e & \text{if } d=e\\ \text{get}-\text{type}(u,e) & \text{else}\end{cases}$$

get−type(share(u,d_1,d_2),d)

$$=\begin{cases}\text{get}-\text{type}(u,d_2) & \text{if } d_1=d\\ \text{get}-\text{type}(u,d) & \text{else}\end{cases}$$

(下转第44页)

(上接第85页)

库系统设计时采用的方法，去构造多媒体数据库系统，必须考虑到由于多媒体数据的引入所产生的各种影响。实践表明，分层设计的思想是开发软件系统的有效方法。利用这种方法，我们将庞大而又复杂的多媒体数据库系统分解为若干层次，下层作为上一层实现的虚拟机，把混杂在一起的系统功能的需求细化到每一层。我们可以自下至上逐步实现各层的功能，从而极大地降低了系统实现的复杂性。用这种方式所设计出来的多媒体数据库系统具有可靠性高、便于移植、易于维护等优点，每层的局部改动对整个系统的影响较小。采用这种思想，我们设计开发了多媒体数据库系统 CDB/M[1]，该系统的成功投入使用，也充分显示了这种分层设计方法的有效性。

参 考 文 献

[1] 周龙骧，柴兴无，分布式多媒体数据库系统的分层体系结构，计算机学报，19(7)1996

[2] M. Muhlhauser, et al., Services, Frameworks, and Paradigms for Distributed Multimedia Applications, IEEE Multimedia, Fall 1996

[3] E. Chang, et al, Reducing Initial Latency in Media Servers, IEEE Multimedia, July-September 1997

[4] T. L. Kunii, et al, Issues in Storage and Retrival of Multimedia Data, Multimedia. Systems, 1995,3

[5] J. H. Friedman, et al, An Algorithm for Finding Best Matches in Logarithmic Expected Time, ACM Trans. on Math. Software,1977,3

[6] S Dao, et al, MB*-Tree: An Index Structure for Content-Based Retrieval, In K. C. Nwosu et al, eds., Multimedia Database Systems, Design and Implementation Strategies, Kluwer Academic Publishers, 1996

[7] N. Beckmann, et al, The R* Tree: An Efficient and Robust Access Method for Points and Rectangles, In Proc. ACM SIDMOD Intl. Conf. on the Management of Data Atlantic City, May 1990

[8] D. V. White, et al, Similarity Indexing with the SS-Tree, In Proc. 12th Intl. Conf. on Data Engineering, New Orleans, Louisiana, February 1996

[9] N. Katayama, et al, The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries, In Proc. ACM SIDMODE Intl. Conf. on the Management of Data, AZ, USA,1997

[10] Chiueh Tzi-cker, Content-Based Image Indexing, In:Proc. 20th VLDB Conf. Santiago, Chile, 1994

[11] T. Bozkaya, et al, Distance-Based Indexing for High-Dimensional Metric Spaces, Conf. Same to [9]

[12] ISO, Hypermedia/Time-based Structure Language: HyTime(ISO 10744). ISO,1992

[13] ISO,Multimedia and Hypermedia Information Coding Expert Group. ISO/IEC JTC1/SC29/WG12,MHEG Working Draft"WD. 1. 0,"Version 1. 0(Feb. 1993)

[14] T. D. C. Little, et al, Synchronization and Storage Models for Multimedia Objects. IEEE Journal on Selected Area in Communications, 8,3(April 1990)

[15] G. A. Schloss, et al, Providing Definition and Temporal Structure for Multimedia Data. ACM Multimedia Systems(1995)3

[16] 巩志国，周龙骧，多媒体对象的 Agent 展示集成模型，软件学报已录用。

[17] R. Barber, et al, Query by Content for Largr On-Line Image Collections, Research Report RJ9408 (82660), June 1993, IBM Research Division, New York

[18] S. Hibino, et al, A Visual Multimedia Query Language for Temporal Analysis of Video Data, Same to [6]

[19] S. C. Kau,et al,MQL-A Query Language for Multimedia Database, In Proc. 2nd ACM Intl. Conf. on Multimedia, ACM Press, 1994

[6] B. J. Kuipers, et al., Higher-order derivative constraints in qualitative simulation. Artificial Intelligence 51, 1991

[7] Giorgio Brajnik et al., Temporal constraints on trajectories in qualitative simulation. In Working Papers of the Tenth International Workshop on Qualitative Reasoning (QR-96), Fallen Leaf Lake, 1996

[8] California. Pierre Fouche & Benjamin Kuipers. Reasoning about energy in qualitative simulation IEEE Transactions on Systems, Man, and Cybernetics 22 (1), 1992

[9] B. J. Kuipers, Abstraction by time-scale in qualitative simulation. In Proceedings of the National Conference on Artificial Intelligence (AAAI-87). Los Altos, CA: Morgan Kaufman, 1987

[10] B. J. Kuipers et al., Using incomplete quantitative information in qualitative reasoning. In Proceedings of the National Conference on Artificial Intelligence (AAAI-88). Los Altos, CA: Morgan Kaufman, 1988

[11] D. Berleant et al., Qualitative-numeric simulation with Q3. In Boi Faltings and Peter Struss (Eds.), Recent Advances in Qualitative Physics, MIT Press, 1992

[12] H. Kay, and B. Kuipers, "Numberical Behavior Envelopes for Qualitative Models" AAAI-93: 1993

[13] H. Kay, "SQSIM: a simulator for imprecise ODE models" University of Texas Artificial Intelligence Laboratory TR AI96-247, March 1996

[14] O. Raiman, "Order of magnititute Reasoning" AAAI-86, 1986

[15] Marcos Vescovi, et al., "Numerical Interval Simulation: Combined Qualitative and Quantitative Simulation to Bound Behaviors of Nonmonotonic Systems" IJCAI-95, 1995

[16] Shen, Qiang and Leitch, Roy "Fuzzy Qualitative Simulation" IEEE Transaction on Systems, Man, and Cybernetics, 23(4) 1993

[17] C. Jack, Schryver "Object-Oriented Qualitative Simulation of Human Mental Models of Complex Systems" IEEE Transaction on Systems, Man, and Cybernetics, 22(3) 1992

[18] K. D. Forbus, et al., "Self-Explanatory Simulations: An Integration of Qualitative and Quantitative Knowledge" AAAI-90, 1990

[19] K. D. Forbus, et al., "Scaling up Self-Explanatory Simulators: Polynomial-time Compilation" AAAI-95, 1995

[20] Kaul Neeraj, et al., "An Artificial Intelligence Approach to Multi-level Mixed-mode Qualitative Simulation of CMOS Ics" Computers Electrotic Engineering, 20(5), 1994

[21] James Paul, et al., "Qualitative Simulation of Electrical and Mechanical Systems" Simulation 63(1), 1994

(上接第34页)

程序段	函数应用
TYPE TT=|T1|T2|T3;	q_0:=init(带参数 T1, T2, T3)
VAR u1, u2, TT;	q_1=create(create(q_0, "u1"), "u2")
t1: T1;	
t2: T2; …	
u1: T1: =t1	q_2: = set − value (set − type (q_1, "u1", T1), "u1", value(t1))
u2: =u1;	q_3=share(q_2, "u2", "u1")
u2: =NIL;	q_4: =set−type(q_3, "U2", t_0)
…	
TYPE u1 WITH uu OF	q_5: =share(create(q_4, "uu"), "u1")
T1: uu: =t1;	if get−type(q_5, "u1")=T1,
u1: T2: =t2	q_6: =set−value(set−type(set − value (q_5, "uu", value (t1)), "u1", value(t2))
|T2: t2: =uu	if get−type(q_5, "u1") = T2, q_6: =q_5(t2=get−value(q_5, "uu"))
ELSE	q_6: =q_5
END	q_7: =set−type(q_6, "uu", t_0)
…	

参 考 文 献

[1] N. Wirth, Design and Implementation of MODULA, Software Practice and Experience, 7, 67～84, 1977

[2] C. A. R. Hoare, An Axiomatic Basis of Computer Programming, Comm. ACM 12(10), 1960

[3] 徐家福, 系统程序设计语言, 北京: 科学出版社, 1983

[4] 周巢尘, 形式语义学引论, 长沙: 湖南科学技术出版社, 1985

[5] B. Meyer, Introduction to the Theory of Programming Language, Prentice Hall, 1990