关于并行语言 Gamma 的结构化扩充*

韦梓楚

(中国科学院 数学与系统科学研究院 数学研究所,北京 100080)

E-mail: zcwei@math08.math.ac.cn

摘要: Gamma(general abstract model for multiset manipulation)语言是一个可以避免人为序化的高层次并行程序设计工具. P. Fradet 等人提出结构化 Gamma 以利于结构化应用数据以及表达特殊的控制约束和程序的自动分析. 在摒弃了基于"地址"的定义方式的基础上,给出结构化 Gamma 的更一般形式的语法、语义描述,在两个层次上讨论结构重集(structured multiset)的类型,并指出,这种新模型保持了有关程序的静态类型检测等理论成果.

关键词: 并行程序设计语言;结构化 Gamma(general abstract model for multiset manipulation);图文法;重集的类型;程序的类型检测

中图法分类号: TP312 **文献标识码**: A

现实世界大量存在的并发行为越来越高地要求程序设计工具可自然和方便地刻画它们. J. P. Banatre 和 D. Le Metayer 比照化学反应模式发展的核心语言 Gamma(general abstract model for multiset manipulation)是一种高度并行的程设模型[1~3],它准确地抓住了并行计算的直觉—— 大批自由交互的原子值的全局性演变,克服了经典语言在处理并行性时往往必须或不自觉地强加人为的顺序性. 其单一的数据对象是重集(multiset,即允许有重复元素的集合),程序是表示反应条件和结果的一组(条件,动作)偶. 程序员可以利用语言十分高级的特征抽象地写出描述各种应用的直观而简练的并行程序,必要时通过程序推导逐步求精. 用 Gamma 编写计算任意一个非空重集 M 的最大元的程序极为简单:

$$\max \quad M$$
$$\text{where } \max \equiv [x, y, x \leqslant y \vdash\!\!> y]$$

这是人的认识的最直接反映:随意比较一对元,抹去小的,不同元对均可同时比较,直至剩下未一元即是最大. 程序应用的对象是任何(基数随意的)有序元素的重集.

然而,原有的 Gamma 模型也存在一些弱点,要害之一是,对程序员来说,语言难以构造数据,不易规定特殊的控制策略,即使人工技巧处理,也不利于程序的自动分析. 有鉴于此,P. Fradet 和 D. Le Metayer(下文略为 F&LeM)近期进而研讨了基于结构重集的结构化 Gamma[4]. 他们把结构重集定义为有数据值关联的、满足某些关系的"地址的集合",形式地刻画了重集和程序的类型,这有益于程序推理,也利于探讨程序的合理性及 Gamma 程序的求精. 然而,地址与存储相关联,不是刻画数据对象的合适抽象. 而且他们的结构重集的定义也未与"结构数据"的一般理念挂上钩,其描述能力较受限制. 本文抛弃地址概念,从结构数据的一般形式出发,重新探讨结构重集及其类型和结构 Gamma 的更一般刻画,例示其强描述能力. 由于结构数据的属性和 F&LeM 的关键成分"关系"存在着关联,因此新方法也保持了该文的理论成果.

1 结构重集(Structured Multisets)

表达数据对象的最好的抽象方式是用其名字. 结构化的数据常用附带某些属性的方式来表示,用域命名其

* 收稿日期: 1999-05-17; 修改日期: 1999-09-13

基金项目: 国家自然科学基金资助项目(69737020);科技部中法先进研究计划基金资助项目(PRA M95-3)

作者简介: 韦梓楚(1941—),男,福建泉州人,研究员,主要研究领域为程序设计方法学.

各属性,如:职工名(年龄,性别,住处,工资,...).不同职工在同一域的值各异.不妨把域名看成是关于对象名的映射,是某类型的函数,用函数形式〈域名〉(〈职工名〉)(而不用习惯的〈职工名〉.〈域名〉)来表示具体职工在该属性域上的取值.

定义 1. 一个结构重集是指结构数据对象(称集元)的集合,记为 M.即 M 的集元可包含若干属性,其形式为〈集元名〉(〈属性〉),而

〈属性〉::=〈属性函数名〉|〈属性函数名〉,〈属性〉

若有必要指出一个具体属性的类型,可在〈属性函数名〉后加":〈类型名〉"表示.当不必强调诸属性时,集元可用其名表示.

定义 2. 结构重集 M 的全体集元名形成其阱集,记为 PM.属性函数把 PM 的成员或者映射为某类型的值,这时称之为值属性函数;或者映入 PM 或 PM 的笛卡尔积,这时它描绘集元间的某种关联,称为关联属性函数.

众集元及其各属性取值共同表达结构重集的全貌,关联属性函数刻画重集的关联结构.结构重集 $\{x_i(\text{val}:\text{int},\text{next}:PM,\text{pter}:\{x_i\})\mid i=1,2,3,4,5,6\}$ 描述一个整型并链指特定元的 6 元链表.$\text{val}(x_i):=i-1$,$\text{next}(x_i):=x_{(3*i \bmod 7)}$,$\text{pter}(x_i):=x_1$ 给出如图 1 所示的结构数据.

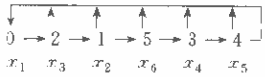

Fig. 1 Linked list by structured multiset
图1 可用结构重集表达的链表

定义 3. 属性值在其值域可引出比较关系:相等关系,序关系,属关系等,称此类式子为属性关系式.

值属性关系式的右边可以是其他集元为变元的一些属性值的函数式(即算术或函数表达式),以 PM 的笛卡尔积为定义域,可写成〈函数名〉(〈集元名序列〉).将这样的函数式作为值赋予某集元的一个属性(需类型一致!)可表示为

〈值属性函数名〉(〈集元名〉):=〈函数名〉(〈集元名序列〉)

对于关联属性关系式,暂只考虑等式情况.因其右边是〈集元名〉或〈集元名元组〉,为简单起见,用关系谓词加以表示,即有

定义 4. 可把关联属性关系等式直接写成〈关联属性函数名〉〈集元名元组〉,元组的第 1 个量理解为等式左边的变元,其他的是关系等式的右边.

此写法把〈关联属性函数名〉看成是一种"关系",它指出一组集元间的某种关联.例如,把属性取值(或赋值)$\text{next}(x_i)=x_j$ 写成关系 next $x_i\ x_j$.我们还可以有

定义 5. 集元及其元组的恒等或交叉关系是指:

eq $x\ y$ iff x 和 y 是同一〈集元名〉

eq $(x_1\ x_2\ \ldots\ x_n)(y_1\ y_2\ \ldots\ y_l)$ iff $n=l\wedge(\forall\ i,\text{eq}\ x_i\ y_i)$

cross $(x_1\ x_2\ \ldots\ x_n)(y_1\ y_2\ \ldots\ y_l)$ iff $\exists\ i,j\ \text{eq}\ x_i\ y_j$

bool 函数是谓词,故 bool 型属性也可看成是一种特殊的关联属性,其取值可写成无关联对象的"关系"形式:

〈bool 属性函数名〉〈集元名〉或 ~〈bool 属性函数名〉〈集元名〉.

2 结构化 Gamma

有结构重集的定义,应重新建立 Gamma 的文法.文献[4]的部分写法被沿用.

2.1 语 法

〈程序〉 ::=〈程序名〉≡[〈反应组〉]

〈反应组〉::=〈反应〉|〈反应〉|〈反应组〉

〈反应〉 ::=〈条件〉⊨>〈动作〉

〈条件〉 ::=〈关系式〉|〈条件〉,〈条件〉

〈关系式〉::=〈关联属性关系式〉|〈值属性关系式〉|〈恒等关系〉|〈交叉关系〉

〈动作〉 ::=〈关联属性关系式〉|〈值属性函数名〉(〈集元名〉):=〈函数式〉|〈集元名〉|〈动作〉,〈动作〉

这种重写规则是一类图文法.〈动作〉定义的后一个〈集元名〉是指未在其反应条件中出现而要新增加的集元.有一个自然的附加条件:〈关联属性关系式〉里的〈集元名〉和〈值属性函数名〉的变元或是出现在对应的反应条件中,或是已在此动作里新指明的;而〈函数式〉里的〈集元名〉只能是已在对应的反应条件中出现的,并且

$$\forall f, f(\ldots v f(x) \ldots) = \perp \text{ if } v f(x) = \perp \quad \text{or} \quad x$$

未出现在对应的反应条件中.

2.2 语 义

一个结构化 Gamma 程序可写成

$$P \equiv [C_1 \models\!> A_1 \parallel C_2 \models\!> A_2 \mid \ldots \parallel C_m \models\!> A_m],$$

其中条件 C_i 是一些

$$r\ x_1\ x_2\ \ldots\ x_l \text{或} r'\ (x_1\ x_2\ \ldots\ x_n)(y_1\ y_2\ \ldots\ y_p) \text{ 或 } v(x)?\ f(x_1, x_2, \ldots, x_q), \tag{1}$$

x_i, y_i 都是集元名. r 是所论重集的集元的关联属性函数名或 eq, r' 是 eq 或 cross, v 是值属性函数名. ? 表示 $=$、$<$、$>$、$\neq$、$\leqslant$、$\in$ 等关系. 动作 A_i 则是一些

$$x \text{ 或 } r\ x_1\ x_2\ \ldots\ x_l \text{ 或 } v(x) := f(x_1, x_2, \ldots, x_n). \tag{2}$$

考虑程序 P 应用于重集 M 上的语义,用 φ 表示语义映射. 对于恒等和交叉关系,其语义如定义 5 所述;对于式(1)的其他关系,按第 1 段所论

$$\varphi[r\ x_1\ x_2\ \ldots\ x_l](a_1, a_2, \ldots, a_l) = \perp \quad \text{if} \quad r(a_1) = \perp$$
$$= -(r(a_1) - (a_2\ \ldots\ a_l)) \text{的真值},$$
$$\varphi[v(x)?\ f(x_1, x_2, \ldots, x_q)](a, a_1, a_2, \ldots, a_q) = \perp \quad \text{if} \quad v(a) = \mid \quad \text{or} \quad f(a_1, a_2, \ldots, a_q) = \perp$$
$$= (f \text{的值类型与} v \text{的类型一致}) \wedge$$
$$(v(a)?\ f(a_1, a_2, \ldots, a_q)) \text{的真值}.$$

若 C_i 有形式 $Z_1, Z_2, \ldots, Z_n$,则定义

$$\varphi[C_i] = \varphi[Z_1] \wedge \varphi[Z_2] \wedge \ldots \wedge \varphi[Z_n].$$

对于式(2),

$$\varphi[x](b) = \{b(\perp, \ldots)\} \qquad /* \text{新集元,其所有属性值暂无定义} */$$
$$\varphi[r\ x_1\ x_2\ \ldots\ x_l](a_1, a_2, \ldots, a_l)$$
$$= \{a_1 \mid_{r(a_1) := (a_2\ \ldots\ a_l)}, a_2, \ldots, a_l\} \qquad /* a_1 \text{的关联属性} r \text{重新赋值,其余不变} */$$
$$\varphi[v(x) := f(x_1, x_2, \ldots, x_n)](a, a_1, a_2, \ldots, a_n)$$
$$= \perp \qquad \text{if } f(a_1, a_2, \ldots, a_n) \text{无定义}$$
$$= \{a \mid_{v(a) := f(a_1, a_2, \ldots, a_n)} \qquad /* a \text{的值属性} v \text{重新赋值,其余不变} */$$

若 A_i 有形式 $Y_1, Y_2, \ldots, Y_k$,则定义

$$\varphi[A_i] = \perp \qquad \text{if } \exists\ j \in [1, k], \varphi[Y_j] = \perp$$
$$= \varphi[Y_1] \cup \varphi[Y_2] \cup \ldots \cup \varphi[Y_k].$$

对每个 i,Gamma 的基本语义是

$$\varphi[C_i \models\!> A_i](a_1, \ldots, a_n, b_1, \ldots, b_k)$$
$$= \perp \quad \text{if } \varphi[C_i](a_1, a_2, \ldots, a_n) = \perp \vee \varphi[A_i](a_1, \ldots, a_n, b_1, \ldots, b_k) = \perp$$
$$= M \quad \text{if } \varphi[C_i](a_1, a_2, \ldots, a_n) \text{不真}$$
$$= M \setminus \{a_1, \ldots, a_n\} \cup \varphi[A_i](a_1, \ldots, a_n, b_1, \ldots, b_k),$$

其中 $\{a_1, \ldots, a_n\} \subseteq M$ 是 C_i 的变元集, $b_1, \ldots, b_k \notin M$ 是在 A_i 中新出现的变元.

于是,结构化 Gamma 程序 P 的语义可以按并行递归结构的语义来看待,一般地说,可以定义为它的按各个反应实现的重写系统的规范式的集合,即

$$\varphi[P]M = \mid \quad \text{if } \forall\ i, \varphi[C_i \models\!> A_i](a_1, \ldots, a_n, b_1, \ldots, b_k) = \perp$$
$$= M \quad \text{if } \forall\ i \in [1..m], \forall\ \{a_1, \ldots, a_{n(i)}\} \subseteq M, \neg\varphi[C_i](a_1, \ldots, a_{n(i)})$$
$$= \bigcup_{i \in [1..m] \wedge \varphi[C_i](a_1, \ldots, a_{n(i)})} \varphi[C_i \models\!> A_i](a_1, \ldots, a_{n(i)}, b_1, \ldots, b_{n(i)})$$
$$/* \{a_1, \ldots, a_{n(i)}\} \subseteq M, b_1, \ldots, b_{n(i)} \notin M */$$

若重写系统无规范式,即按 P 重写时系统发散,仍不妨用并行递归程序的流语义来理解.

2.3 一个例子

可用这样的结构 Gamma 描述复杂问题. 考虑人类社会发展演化的一个执行一代亲缘禁婚和实施计划生育的简化模型. 人类社会的基本对象是下述形式的集元,社会是由一簇这样的集元而形成的重集 Society:

person(age: N, sex: bool, marry_bear: bool, birthday: ([1..12], [1..31]), parents: PM×PM).

属性分别表示年龄、性别、婚育状况、生日和双亲.简化模型可用下述 Gamma 程序给出:

```
SC≡[age(x)>25,age(y)∈[age(x)-3,age(x)+5],sex(x)≠sex(y),
    ~marry_bear(x),~marry_bear(y),~cross parents(x) parents(y)
    ⊨> z,marry_bear(x),marry_bear(y),age(z) :=0,sex(z) :=rendom,
      marry_bear(z) :=false,birthday(z) :=rendom1,parents(z) :=(x,y)
    ‖ birthday(x)= clock  => age(x) :=age(x) +1
    ‖ age(x)>90 ⊨> ]
```

有 3 个并发的反应,为首的表示任何一对足龄而相近、未婚、非一代亲缘的异性男女可结婚且生育一个后代,此子女的性别和生日由随机函数给出;次一个指出按时钟(clock)凡达到生日者即增龄;末一个反应表达超过 90 岁者消亡.任意给出社会成员的一个初始重集 Society,可执行 SC Society 来观察社会的演化结果.

2.4 程序正确性问题

如何判定一个结构化 Gamma 程序正确与否?回想函数式程序设计把程序看成函数,程序的正确性一般依赖于推理和论证,但其基础是:程序对作用数据在类型上要符合某种转换规则,或即维持类型不变性.结构化 Gamma 程序包含并行递归结构,它也应当维持一定的类型不变性.为此,先来讨论结构重集的类型.

3 结构重集的全类型和构造性类型

说起正整数类型 N,立即会想到是指 1,2,3,4,... 这些数,以及所有后面的数是其前一个数加 1 这种关系和由此引出的序关系与一些运算规则.因而可借助基说明 bas 和后继关系 ass 构造性地把类型 N 看成(bas 1)及由它构造出的数据群 ass 1,ass(ass 1),ass(ass(ass 1)),...,或者说

```
N=Numb x
Numb x=(bas 1) x                    /*右边即 eq x 1/*
Numb x=ass y x,Numb y
```

类似地,也可引入类型概念来刻画结构化重集,借助基于重集的关联关系的重写规则来描述.结构重集的定义蕴示其类型的两重含义:集元的诸值属性函数的类型反映出的整体值类型;集元的关联属性表现出的重集的构造性.

3.1 语法和语义

结构重集的类型用下述文法来定义:

```
〈类型说明〉::=〈类型名〉=〈产生式〉[〈非终结项〉=〈产生式〉]*
〈产生式〉  ::=〈参量〉|〈关联关系式〉|〈非终结项〉|〈产生式〉,〈产生式〉
〈非终结项〉::=〈类型构造名〉〈参量组〉
〈参量组〉  ::=〈参量〉|〈参量〉〈参量组〉
〈参量〉    ::=〈集元名〉|〈集元名〉(〈属性〉)
〈集元名〉  ::=〈变量名〉|〈常名〉
```

这里的关联关系式就是上节的关联属性关系式,可表示对集元的关联属性的赋值.〈属性〉前面已有说明.由于一个重集的各集元的属性函数名序列划一,故只需对第 1 个出现的集元加属性说明.为了认读方便,对产生式可适加括号.

定义 6. 称一个结构重集为属于某类型是指它可以完全由指定的重写系统产生.各值属性函数名都给出明确的类型说明的类型定义称为全类型定义;忽略对值属性的类型规定,只考虑结构重集完全由与关联属性相关的重写系统产生的类型称为构造性类型.

作为一个例子,整型表单(list)类型可定义为

```
List(int)=L x(val:int,next)
  L x=(x,next x y),L y
  L x=(x,next x nil).
```

定义泛类型表单,只需首行改为 List=L x(val,next).

下文侧重后一层意义下的类型的讨论.此时,本文的语法与文献[4]中用地址间的关系定义类型的语法相

近,我们也可把类型 T 的定义与一个会产生 T 类型的任何重集的结构化 Gamma 程序(记成 Gen^T)联系起来,相当于把=号看成⊨>,规则间用‖号相连,把〈类型构造名〉看成特殊的关联属性关系,类型定义中的非终结项和相关程序中的关联关系用同样的记号表示.与 List 类型联系的结构化 Gamma 程序可定义为

$$\begin{aligned}Gen^{List} \equiv [&List \models> L\ x(val,next)\\ &\| L\ x \models> (x,next\ x\ y), L\ y\\ &\| L\ x \models> (x,next\ x\ nil)]\end{aligned}$$

此程序应用到只含有原子 List 的重集可产生一切有穷表单,惟有其元素的值属性待定.为了明确生成有穷整型表单,只需将首行改成

$$Gen^{List(int)} \equiv [List(int) \models> L\ x(val:int,next).$$

定义 7. 重集 M 的抽象是指忽略其各集元的各个值属性的具体化,记为 $|M|$.

定义 8. 一个重集 M 是属于类型 T 的,记成 $M:T$,是指 $\{T\} \longrightarrow_{Gen^T} |M|$.

Gen^T 的逆也是个重写系统,记为 $\rightsquigarrow_T$.

定义 9. $M \longrightarrow_{Gen^T} M' \Leftrightarrow M' \rightsquigarrow_T M$. 自然有下面的命题 1.

命题 1. 重集 M 属于类型 T iff $|M| \rightsquigarrow_T \{T\}$.

3.2 类型之例

F&LeM 提到的函数式语言中的一些抽象类型都不难用本模型定义,用于表达图结构尤其具有简明之处.如,双向链接表单的类型是

$$\begin{aligned}DbList = &DL\ x(value,next,pred)\\ &DL\ x = (x,next\ x\ y),(DL\ y,pred\ y\ x)\\ &DL\ x = x,next\ x\ nil\end{aligned}$$

全与末元素链接的表单类型是(如图 2 所示)

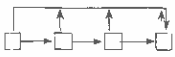

Fig. 2 A list with all elements linked to the last one
图2 全链接末元的表单

$$\begin{aligned}Listlast = &LL\ x(value,next,last)\ z\\ &LL\ x\ z = (x,next\ x\ u,last\ x\ z),LL\ u\ z\\ &LL\ z\ z = z,next\ z\ nil,last\ z\ z\end{aligned}$$

二叉树的类型是

$$\begin{aligned}Bitree = &Bt\ x(value,left,right)\\ &Bt\ x = (x,left\ x\ u,right\ x\ v),Bt\ u,Bt\ v\\ &Bt\ x = x,left\ x\ nil,right\ x\ nil\end{aligned}$$

而叶子串连的二叉树(如图 3 所示)对应的类型则是

$$\begin{aligned}Btreelinked = &Btl\ x(value,left,right),x_l,x_r\\ Btl\ y\ u\ v &= (y,left\ y\ l,right\ y\ r),(Btl\ l\ u\ z,right\ z\ w),Btl\ r\ w\ v\\ Btl\ y\ u\ v &= (y,left\ y\ u,right\ y\ r),(u,left\ u\ nil,right\ u\ w),Btl\ r\ w\ v\\ Btl\ y\ u\ v &= (y,left\ y\ l,right\ y\ v),(Btl\ l\ u\ z,right\ z\ v),(v,left\ v\ nil,right\ v\ nil)\\ Btl\ y\ u\ v &= (y,left\ y\ u,right\ y\ v),(u,left\ u\ nil,right\ u\ v),(v,left\ v\ nil)\end{aligned}$$

与前面的例子相比,从这里的第 2 行起谓定义式的写法更加灵活,变量 y,u,v 可分别用第 1 行的变量 x,x_l,x_r 来通代.

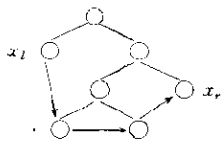

Fig. 3 Binary tree with linked leaves
图3 叶子串连的二叉树

环形链表的类型是(如图 4 所示)

$$\begin{aligned}CircList = &CL\ x(value,next)\ x\\ &CL\ y\ z = (y,next\ y\ u),CL\ u\ z\\ &CL\ y\ z = y,next\ y\ z\end{aligned}$$

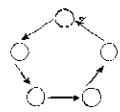

Fig. 4 Circle list
图4 环形链表

这里避免了 F&LeM 的文章中用增加特殊记号来指示变量可能相同这样一种麻烦.总而言之,凡在该文的模型中能描述的类型按本文的模型都没有表示上的困难.第 2.3 节例子中的 Society 的类型可定义为

$$\begin{aligned}&Type\ Society(S_0) = Memb\ x\\ &Memb\ x = Member\_of(S_0)\ x\\ &Memb\ x = (x,parents\ x\ y\ z),(Memb\ y,sex\ y),(Memb\ z,\tilde{}sex\ z)\end{aligned}$$

Member_of(S_0)是个关系函数,S_0 是基始集.

3.3 带类型的 Gamma 程序设计

现在,我们可以像函数式程序设计中所做的那样,把任何结构化 Gamma 程序看成属于一定的类型,即它能应用于什么类型的重集上.置换排序(如 bubble_sorting 等)有如下简单表示:

$$\mathrm{dSorting}:\mathrm{List}\equiv[\mathrm{next}\ x\ y,\mathrm{val}(x)>\mathrm{val}(y)\models>\ \mathrm{val}(x):=\mathrm{val}(y),\mathrm{val}(y):=\mathrm{val}(x)].$$

可借助类型表达某种控制策略.如着眼实效作顺序排序,可先定义重集的类型:

$$\begin{aligned}
\mathrm{List}_0 &= L_0\,x(\mathrm{val},\mathrm{hd}:\mathrm{bool},\mathrm{next},\mathrm{pter})\\
L_0\,x &= (x,\mathrm{next}\ x\ y),L_0\,y\\
L_0\,x &= L_1\,x\,z,\mathrm{hd}\ x,\mathrm{pter}\ x\ z\\
L_1\,x\,z &= (x,\mathrm{next}\ x\ y),L_1\,y\,z\\
L_1\,x\,z &= L_2\,z\\
L_2\,x &= (x,\mathrm{next}\ x\ y),L_2\,y\\
L_2\,x &= x,\mathrm{next}\ x\ \mathrm{nil}
\end{aligned}$$

程序写成

$$\begin{aligned}
&\mathrm{SeqSorting}:\mathrm{List}_0\equiv\\
&[\mathrm{hd}\ x,\mathrm{pter}\ x\ y,\mathrm{val}(x)>\mathrm{val}(y)\models>\ \mathrm{val}(x):=\mathrm{val}(y),\mathrm{val}(y):=\mathrm{val}(x)\\
&\ \|\ \mathrm{hd}\ x,\mathrm{pter}\ x\ y,\mathrm{next}\ y\ z,\mathrm{val}(x)\leqslant\mathrm{val}(y)\models>\ \mathrm{pter}\ x\ z\\
&\ \|\ \mathrm{hd}\ x,\mathrm{pter}\ x\ y,\mathrm{next}\ x\ z,\mathrm{next}\ z\ w,\mathrm{next}\ y\ \mathrm{nil},\mathrm{val}(x)\leqslant\mathrm{val}(y)\models>\ \mathrm{hd}\ z,\mathrm{pter}\ z\ w]
\end{aligned}$$

事实上,List_0 是 List 的一个精细化,增加的数据结构给程序补充了新的、便于描述的控制手段,即利于程序的求精.

4 类型检查问题

由于我们的描述方法同 F&LeM 使用的"关系"模型存在着关联,文献[4]中的一些理论结果都能在本模型下得到类推.前文提过,结构化 Gamma 程序应当维持其应用数据的类型不变,即程序的应用数据和结果数据保持类型一致.这就提出了与程序相关的类型检查问题,实际上就是要讨论 $C\models>A$ 形式的反应的类型不变性的检测.

定义 10. 反应 $C\models>A$ 是关于 T 良类型的是指 $\forall\ M:T,((C\models>A)M):T$.

检查可分为两步:

(1) 常规检查反应条件 C 中的值属性关系式和反应动作 A 中的赋值式符合 T 的类型定义(的首变量)规定的各值类型;

(2) 检测 $C\models>A$ 的非数值部分是否保持 T 的构造性类型(除数值性说明外,T 的形式完全由它确定,故仍用同名 T 称之).问题只涉及到一些关系,完全可类同于文献[4]中的静态类型检测方法.P.Fradet 等人已指出,对于用非受限上下文无关图文法描述的类型,这样的性质一般是不可判定的,但对此文法的一些子类则存在完备的检查算法.对于绝大多数实际应用,给出一种分析算法已足够了.我们的情形相同,算法可类似推导,这里只列出结果.

把经第(1)步检查后去掉数值部分而剩下的条件和动作分别记成 $|C|_{S(C)}$ 和 $|A|_{S(A)}$.$S(C)$ 和 $S(A)$ 分别表示它们的包括数值部分在内的变元集.我们要检测的是

$$\forall\ M:T,((|C|_{S(C)}\models>\ |A|_{S(A)})M):T.$$

构造性类型检测算法:

$$\begin{aligned}
&\mathrm{Typecheck}\ (P,T)=\\
&\forall\ (|C|\models>\ |A|)\ \mathrm{of}\ P,\ \mathrm{if}\ S(A)=S(C)\wedge|A|=(\text{空})\\
&\qquad\qquad\qquad\qquad\mathrm{then\ True}\\
&\qquad\qquad\qquad\qquad\mathrm{else\ Check}\ (|A|,T,\mathrm{Build}(|C|,\{|C|\},T))
\end{aligned}$$

T 是程序 P 所应用的数据(即一个重集)的类型.Check,Build 等是文献[4]中定义的一些函数过程.许多反应式不涉及修改关联关系,一个条件判断可提效.我们也有下面的命题.

命题 2. 对于 P,若 $M1:T,(M1\rightarrow_P M2 \wedge \text{Typecheck}(P,T))$则 $M2:T$.

对于 dSorting,只有一个 $C \models> A, S(C)=S(A)=\{x,y\}$,并且 $|A|=$〈空〉,此程序应用数据的类型(list)不变.事实上,从动作部分知道,除次序外,它保持数值集不变.反应结束后,条件$(\forall x,y \text{ next } x\ y)\ \text{val}(x)>\text{val}(y)$不成立,因此,被 dSorting 的重集自然是由小到大排好序的.SeqSorting 有 3 个反应式,第 1 个同理,保持数据类型不变,后两个也可用算法函数 Check 检测出来.

5 结 语

Gamma 语言是相当高级的并行程序设计工具,纯 Gamma 应用于数值型重集很方便,为了讨论在有数值间关联的重集上的应用并利于自动分析,则需结构化 Gamma 及相关的结构重集.本文意在排除 F&LeM 的"地址"概念,探讨结构化 Gamma 和结构重集的最一般刻画.地址的重集,其元素是单值的,除横向关系外不会有纵向层次.本文基于一般结构数据作对象的重集模型,对象(此处即重集元素)可以有多属性值且诸属性可以有纵向层次,可以考虑不同层次间的复杂关联关系.文献[4]的结构化 Gamma 反应的语义以地址关连的值和关系为单位,导致程序各反应的动作部分必然有许多书写重复.在我们的模型下,反应的语义以对象为单位——反应后抹去在动作部分不再出现的重集元,可避免重写条件部分已有而不被修改的关系.F&LeM 把地址 x 及其值 $\bar{x}$ 分开表示,对重集实际上只描述其构造性类型,我们的模型对重集的类型则可作完全的讨论.

Gamma 语言的创立者提出"重集"来表达其结构数据,这似源于纯 Gamma 的反应主要面向许可多出现的数据值,进入结构化 Gamma,各数据值只是对象的某侧面,而对象是被区分的,"重集"概念不是必用的——结构化 Gamma 应用的结构数据就是有相同数据结构的对象的"集合".

References:

[1] Banatre, J. P., Le Metayer, D. A new computational model and its discipline of programming. INRIA Research Report 566, France, 1986.

[2] Banatre, J. P., Le Metayer, D. The gamma model and its discipline of programming. Science of Computer Programming, 1990,15(1):55~77.

[3] Huang, Lin-peng, Tong, Wei-qin, Ni, De-ming, *et al*. An introduction to parallel computation model GAMMA. Computer Science, 1994,21(5):20~24 (in Chinese)

[4] Fradet, P., Le Metayer, D. Structured gamma. Science of Computer Programming, 1998,31(2,3):263~289.

附中文参考文献:

[3] 黄林鹏,童维勤,倪德明,等.并行计算模型 Gamma 概述.计算机科学,1994,21(5):20~24.

On the Extension of Structured Gamma

WEI Zi-chu

(*Institute of Mathematics, Academy of Mathematics and System Science, The Chinese Academy of Sciences, Beijing* 100080, *China*)

E-mail: zcwei@math08.math.ac.cn

Received May 17, 1999; accepted September 13, 1999

Abstract: The Gamma formalism is a high level parallel programming tool without artificial sequentiality. P. Fradet and D. Le Metayer proposed recently a model of structured Gamma which makes it easy for the programmer to structure the data and to specify particular control strategies. It also makes the automatic analysis of programs easier. Without relying on the "address" concept used by them, a more general form of the structured Gamma is proposed and its syntax and semantics are described in this paper. Types of structured multisets have been discussed at two levels. The author also shows that the new definition keeps the theoretic results, such as static type checking of programs untouched.

Key words: parallel programming language; structured Gamma; graph grammar; type of multiset; type checking of program