基于 CCS 执行模型的逻辑式语言 POLYLOG 的设计与实现

徐凯　章萃

(复旦大学计算机科学系，上海 200433)

DESIGN AND IMPLEMENTATION OF THE NEW LOGIC PROGRAMMING LANGUAGE POLYLOG BASED ON THE CCS EXECUTION MODEL

Xu Kai and Zhang Cui

(Computer Science Department, Fudan University, Shanghai 200433)

ABSTRACT

In this paper, we have solved two problems in logic programming language, i.e., the selection of parallel execution model and the trade off between running efficiency and using flexibility. We proposed a new comprehension on "Algorithm=Logic+Control", designed and implemented a new programming language POLYLOG. In POLYLOG, the concept of meta control is introduced. The CCS Execution Model is taken to be the internal implied meta control mechanism, and the Relation Type is taken to be the external explicit meta control facility. The parallelism analysis is finished at the compilation time.

摘　要

本文解决了逻辑式程序设计语言中的二个问题：并行执行模型的选择、语言的执行功效与其使用方便灵活之间的权衡. 我们提出了对 Algorithm=Logic+Control 的新理解,

1989 年 10 月 5 日收到， 1990 年 7 月 12 日定稿. 本文得到国家科学基金会资助. 作者 徐凯，1987 年在上海复旦大学获硕士学位，现工作单位为上海计算机软件实验室，任工程师，目前主要研究领域为计算机人工智能、计算机语言。 章萃，女， 37 岁， 1988 年在南京大学获博士学位，现任上海复旦大学教授，主要研究领域为计算机人工智能、计算机语言.

© 中国科学院软件研究所　http://www.jos.org.cn

设计并实现了新型逻辑式语言 POLYLOG. 在 POLYLOG 中，我们引进了元级控制的概念， CCS 执行模型作为内部隐含的元级控制机制，关系类型作为外部显式的元级控制设施. 程序的并行性分析在编译阶段完成.

§ 1. 引　言

逻辑式语言是一种新型程序设计语言，本文研究如何通过开发与并行性来提高它的执行功效. 经过对现有与并行开发策略的研究和渐近时空复杂度的分析 [9][14], 我们以静态确定关系 I/O 、主动求值的方法作为开发与并行的基本策略，以此为出发点，重点解决两个问题，一是计算模型的选择，二是如何解决语言的高功效与语言使用灵活性之间的矛盾.

为了统一解决上述两个问题，我们引进了元级控制的思想，把 Kowalski 提出的 Algorithm=Logic+Control 扩展理解为 Algorithm=Logic+Control=Object_ knowledge+Meta_ knowledge+Control[22], 并设计了 POLYLOG 语言系统.

Meta_knowledge 是指系统外部的显式的元级控制设施在 POLYLOG 中体现为关系类型、数据类型. 多形关系既能给用户一定的灵活性，又能使关系的 I/O 静态确定，这样，子句中各关系计算的半序性完全静态确定，无需任何动态并行性的分析开销.

Control 是指系统内部的元级控制设施，隐含于系统的实现中，在 POLYLOG 中体现为 CCS 执行模型 [4][7][8][9], 借此来保证语言系统的功效性，此模型现处于理论阶段.

对 Object_knowledge, 语言的编译将同时完成元级设施的处理及程序的并行性分析，并生成反映并行性的 CCS 代码.

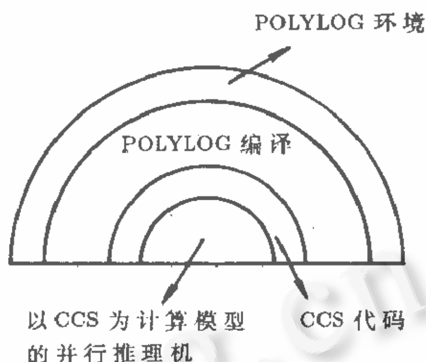

图 1

POLYLOG 系统用 Prolog 写成，其抽象结构如图 1 所示.

§ 2. CCS 模型的并行执行单位

2.1 CCS 模型描述及分析

CCS 模型是以信道段 CCS 为并行执行单位的与并行执行模型 [4], 这个模型减少了推理器之间的通讯，简化了并行执行中的动态控制，充分开发了子句内各关系的并行性.

这里，我们沿用 [4] 中的记号 (设子句为 R: $-R_1,\cdots,R_n$).

关系 R_i 、 R_j 关于变量 X 数据相关记为 $\langle R_i,R_j\rangle$, X 称为 R_i 给 R_j 的信件.

由于关系的 I/O 静态确定，故子句中 $R_1,\cdots,R_n$ 依数据依赖关系形成一个半序，从而构成子句的关系拓扑图.

R_i 为 CCS 始点，如果它满足如下条件之一：

1. $1M(R_i)=0$

© 中国科学院软件研究所 http://www.jos.org.cn

2. $1M(R_i) > 1$

3. $1M(R_i) = 1$, 但存在 $\langle R_t, R_i \rangle$, $OM(R_t) > 1$, 且存在 $k \neq i$, $\langle R_t, R_k \rangle$, length(Rk)=Max{ length(R_j)|$\langle R_t, R_j \rangle$}, R_t 选择 R_k 作为其在 CCS 中的后继点.

R_i 为 CCS 终点，如果它满足如下条件之一：

1. $OM(R_i) = 0$

2. $OM(R_i) = 1$, 设为 $\langle R_i, R_k \rangle$, 但 $1M(R_k) > 1$.

上述定义中

$$\mathrm{length}(R_i) = \begin{cases} 0, & \text{if } OM(R_i) = 0 \\ \mathrm{Max}\{(\mathrm{length}(R_t) | \langle R_i, R_t \rangle\} + 1, & \text{if } OM(R_i) <> 0 \end{cases}$$

2.2 信道段划分算法

算法 2 – 1 给出怎样从 CCS 始点出发，找出整个 CCS.

算法 2 – 2 给出怎样找到子句的关系拓扑图的所有 CCS.

算法 2 – 1 并没有采用函数 length, 而是采用了广度优先搜索算法寻找最大 CCS.

```
ALGORITHM 2 – 1: FindCCS (G, Start, CCS);
Input: topol graph G(VSet, ESet);
   Start< –V
Output: a CCS beginning with Start.

BEGIN
  Path: =[Start];
  PathList: =[Path];
  SegmentSet: =φ;

  While PathList<>[ ]do
    begin
      Path: =CAR[PathList];
      PathList: =CDR[PathList];
      assume Path=[Vn, ···,V1];
      SuccSet: ={V|IM(V)=1 && V∈VSet && (Vn, V) ∈ ESet };
        /* if IM(V)=0 or IM(V)>1, then V is certainly a start node */
      if SuccSet=φ then SegmentSet: = SegmentSet∪{Path}
      else For V ∈ SuccSet do
        begin
          NewPath: =[V, Vn, ···, V1];
          PathList: =append(PathList, [NewPath]);
        end;
    end;
  take a Segment ∈ SegmentSet where Segment has the maximum length;
  produce the path CCS according to Segment and Eset;
END;

ALGORITHM 2 – 2 CCSSet(G, CCSSet);
Input: topol graph G(VSet, ESet);
Output: all CCS in G (CCSSet).
```

© 中国科学院软件研究所 http://www.jos.org.cn

```
BEGIN
  StartSet: ={V|IM(V)=0};
  CCSSet: =φ;
  While StartSet<> φ do
    begin
      take Start in StartSet;
      StartSet: = StartSet − Start;
      call FindCCS(G, Start, CCS);
      CCSSet: =CCSSet∪{CCS};
      NewStart: ={V| < ⟨W, V⟩ ∈ESet && W∈CCS && V∉ any CCS in CCSSet };
                                    /* IM(V)>=1 */
      StartSet> =StartSet NewStart;
    end;
END;
```

从 CCSSet 出发，我们可以构造出子句的以 CCS 为结点的 CCS 拓扑图.

§ 3. POLYLOG 语言

3.1 POLYLOG 语言及其元级控制设施

在标准逻辑式语言中并无数据类型的概念，在 POLYLOG 中，我们提供了数据类型的元级控制设施，用于检查数据项匹配的合法性； POLYLOG 中另一元级控制设施是关系类型.

定义 3-1: 设 R 是有 n 个参数的关系 ($n\geq0$), 每个参数有二个 I/O 状态——输入 (i) 和输出 (o), 那么 R 的 n 个参数 I/O 状态的所有组合构成的集合称为 R 的多形，记为 POLY(R), 其中的每个元素称为 R 的一个形态.

定义 3–2: 设 $d_1\cdots d_n$ 是一个数据类型序列， $POLY_n$ 是一个元素的分量个数为 n 的多形，那么 $(d_1,\cdots,d_n)POLY_n$ 称为一个关系类型. 若 $|POLY_n|=0$ 或 1, 则称它为单一关系类型；若 $|POLY_n|>1$, 则称它为多形关系类型.

称 $R(d_1,\cdots,d_n)POLY_n$ 为一个关系说明.

定义 3–3: 关系的定义子句中，对多形关系指出的形态称为该关系的调用形态，其集合称为仍用形态集.

在实际运用中，可用 ? 表示一个参数同时具有 I 和 O 两种状态，如 (i, i, ?) 的 POLY={(i, i, i), (i, i, o)}.

3.2 一个例子

本文不对 POLYLOG 文法作详细讨论，这里仅给出一个例子说明.

```
Types
  list=integer*
  list_of_list=list*

Relations
  append(list, list, list)      (i, i, o)   (i, o, i)
  alllist(list_of_list, list)     (i, o)
Clauses
```

© 中国科学院软件研究所　http://www.jos.org.cn

```
append([ ], , ,).
append([X|L1], L2, [X|L]): -append(L1, L2, L).

alllist([ ], [ ]).
alllist([L], L).
alllist([L1, L2|LL], Lout): -
  (i, i, o) append(L1, L2, L3),
  alllist([L3|LL], Lout).
```

§ 4. POLYLOG 语言编译的设计与实现

POLYLOG 的编译完成如下几点： 1. 对 POLYLOG 程序进行语法分析； 2. 完成对元级控制设施的处理； 3. 完成对 POLYLOG 程序的并行性分析； 4. 生成为 CCS 执行模型所接收的代码. 本文重点介绍关系类型的语义处理及 POLYLOG 子句的并行性分析.

4.1 关系类型的语义分析

在 POLYLOG 内部，对多形关系 R, 我们从 POLY 中找出 R 的所有有效 I/O 形态，并用有效单一形关系集合等价地代替多形关系 R, 从而达到在内部关系的 I/O 形态唯一确定的目的.

定义 4–1: 设 FLOW∈POLY(R), (FLOW 是 I/O 确定的), 那么称 FLOW 是 R 的有效形态，如果对 R 的每一定义子句满足：

1. 子句中不存在变量的 I/O 冲突和类型冲突；

2. 除了调用形态为 FLOW 的 R 的调用之外，对任何调用关系 R_i, 存在确定 $FLOW_i$, $FLOW_i$ 是 R_i 的有效形态.

若 FLOW 是 R 的有效形态，则记 R[FLOW] 为 R 的有效单一形关系.

在此语义分析阶段， POLYLOG 编译生成一信息表：

RelationTab

Name	parm type list	min set of flows	Set of valid flows	Set of invalid flows
R	TypeL	POLY	VF	NVF

其中有效形态集 (VF) 和非有效形态集 (NVF) 将在程序的并行性分析时求出.

4.2 POLYLOG 程序的并行性分析

程序的并行分析解决如下问题：

1. 找出多形关系的所有有效形态，最终用有效形态关系集代替多形关系.

2. 将子句中对多形关系的调用改为对相应的单一关系的调用.

3. 在子句内各关系调用确定之后，分析各关系计算的半序性，生成中间结构表示以 CCS 为单位的计算次序，以便在代码生成阶段生成 CCS 代码.

以上问题关键在于如何确定子句中诸关系的调用形态. 一旦该问题得到解决，那么建立一定的替代机制，我们就可用有效关系解决 1 、 2 二个问题；用子句的变量定义引用表 (REF-DEF Table), 就可得到关系计算半序图，划分 CCS, 进而得到 CCS 半序图，方便地生成中间结构.

© 中国科学院软件研究所 http://www.jos.org.cn

事实上，定义 4–1 给出了解决该关键问题的方法.

为此我们采用了 SelectTag 算法，并依靠 PROLOG 的回溯机制，得到子句内关系的所有可能的调用形态的组合.

```
ALGORITHM 4-1: SelectTag(INFList, RelationTab, DEFList);
Input: INFList=[INF_1, ···, INF_n]
       where INF_i is indefinite flow of ith call in clause,
       RelationTable.
Output: DEFList=[DEF_1, ···, DEF_n]
        where DEF_i is definite flow of ith call in clause.
BEGIN
  for i:=1 to n do
    if INF_i is definile
    then DEF_i:=INF_i
    else begin
           assume R is the ith relation call;
           assume INF_i is equal to flow set FlowSet;
           DEF_i:=Flow where Flow ∈ FlowSet
                        && Flow ∈ RelationTab[R].POLY
                        && Flow ∈ RelationTab[R].NVF;
         end;
END;
```

算法 4–2 确定每个多形关系的形态；算法 4–3 检查 FLOW 是否为 R 的有效形态.

```
ALGORITHM 4-2: DeterminePOLYProgram(POLYProgram, RelationTab, MONOProgram);
Input: POLYLOG source program POLYProgram;
       RelationTab.
Output: MONOProgram (with definite I/O);
        RelationTab modified.
BEGIN
  ClauseL:=[ ];
  For R ∈ POLYProgram do
    For Flow ∈ RelationTab[R].POLY do /* Flow is of definite I/O */
        DeterminePOLY(R, Flow, RelationTab, ClauseL);
  produce MONOProgram;
END;

ALGORITHM 4-3: DeterminePOLY(R, FLOW, RelationTab, ClauseL);
Input: relation R,
       Flow is a definite flow of R;
       RelationTab;
       ClauseL.          /* ClauseL is for whole program */
Output: check whether Flow is a valid flow or not;
        RelationTab modified;
        ClauseL modified.
BEGIN
  if Flow ∈ RelationTab[R].NVF then return FALSE;

  if Flow ∈ RelationTab[R].VF  then return TRUE;
```

© 中国科学院软件研究所 http://www.jos.org.cn

```
RelationTab[R].VF:=RelationTab[R].VF∪{FLOW};

if for all clause of R /*assume it is R:-INF1 R1,···, INFn Rn.*/
   there is a definite clause R: -F1 R1,···,Fn Rn /* using SelectTag */
   which statisfies 1. no conflit about I/O and type in
                       REF-DEF table of its variables;
                    2. for all i(1 <= i <= n ), /* recursive call */
                       DeterminePOLY(Ri, Fi, RelationTab, ClauseL)=TRUE
then begin
        produce all these definite clauses in NewClauseL;
        ClauseL:=append(ClauseL, NewClauseL);
        produce new relation R' with R and definite I/O Flow;
        produce new clauses of R' with ClauseL;
        return TRUE;
     end
else begin
        RelationTab[R].VF:=RelationTab[R].VF-{Flow};
        RelationTab[R].NVF:=RelationTab[R].NVF∪{Flow};
        return FALASE;
     end;
END;
```

本文以关系的 I/O 静态确定、主动求值为方案，研究了逻辑式语言的与并行开发问题，整个工作可用图 2 表示.

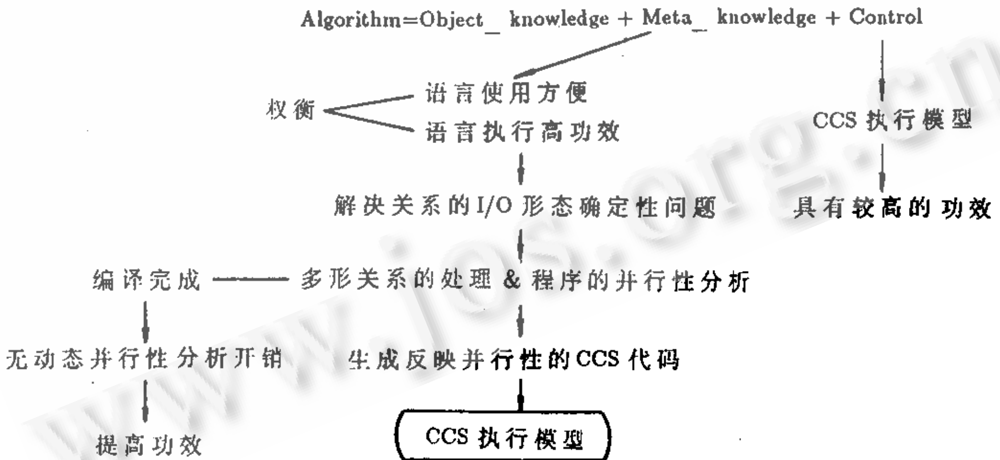

图 2

在研究工作中，我们认为设计并实现 CCS 模型的实际推理机是十分重要的，这将是我们进一步研究方向之一.

© 中国科学院软件研究所 http://www.jos.org.cn

参考文献

[1] Gallaire, H. & Lasserre, C., Metalevel Control for Logical Programs, Logical Programming Edited by K. L. Clark and S. -A. Tarnlund.

[2] Dincbas, M. & Pierre Le Pape, J., Metacontrol of Logic Programs in METALOG, Proceeding of the International Conference on Fifth Generation Computer Systems, 1984.

[3] Zhang Cui, Zhao Qinping & Xu Jiafu, Kernel Language KLND, Journal of Computer Science and Technology, Vol. 1, No. 3, 1986.

[4] 章萃， CCS: 一个新的逻辑式程序设计语言的并行执行模型，计算机研究与发展， 1991. 7.

[5] 管纪文，元知识及其在专家系统中的运用，计算机科学， 1987. 1.

[6] Clark, K. L. & Gregory, S., PARLOG: Parallel Programming in Logic, Imperial College, Reserch Report, Doc. 84/4, 1984.

[7] Conery, J. S., The AND/OR Model for Parallel Interpretation of Logic Programs, Ph. D. Th., UC Irvine, 1983.

[8] Ciepielewski, A., Towards a Computer Architecture for OR-Parallel Execution of Logic Programs, Ph. D. Th., Royal Institute Technology, 1984.

[9] 章萃，并行推理机系统 NDPIS 中 KLND 抽象机 KLND-AM 的设计与分析，南京大学博士学位论文， 1986. 10.

[10] Clark, K. L., IC-Prolog Language Features in Logic Programming, Academic Press Inc., London, 1984.

[11] Shapiro, E., Object Oriented Programming in Concurrent Prolog, New Generation Computing, No. 1, 1984.

[12] Wisee, W. J., Epilog-Prolog, Data Flow Arguments for Combining Prolog with a Data Driven Mechanism, Sigplan Notices, V17-12, Dec. 1982.

[13] 徐家福、章萃、赵沁平，并行推理系统 NDPIS 的设计，计算机研究与发展， 1988. 3.

[14] 章萃，并行推理策略的渐近时空复杂度分析及其应用，计算机学报， 1988. 5.

[15] Goto, A. et al. Highly Parallel Inference Engine PIE-GOAL Rewriting Model and Machine, New Generation Computing, Vol. 2, No. 1, 1984.

[16] 章萃，抽象机 KLND-AM 的并行性开发与软件格局设计，计算机研究与发展， 1988. 6.

[17] DeGroot., Restricted AND-Parallelelism, Proceedings of the International Conference on FGCS, 1981.

[18] DeGroot, D., Restricted AND-Parallelism and Side Effect, Proceedings of 4th IEEE Symposium on Logic Programming, 1987.

[19] Floyd, J. W., Fundation of Logical Programming, Springer-Verlag, Berlin Heideberg, New York, Tokyo, 1984.

[20] Kowalski, R., Logic for Problem Solving, North-Holland, Amsterdam, 1979.

[21] 陈禾，徐凯，语法分析程序的自动生成，计算机软件行业协会青年协会第一届学术会议， 1988. 5, 北京.

[22] 徐凯，章萃，逻辑程序设计语言中的元级控制，知识工程进展，中国地质大学出版社， 1988.

[23] Clocksin, W. F. and Mellish, C. S., Programming in Prolog, Springer-Verlag, Berlin, 1981.

[24] Campbell, J. A., Inplementations of Prolog, Ellis Horwood Limited, 1984.

© 中国科学院软件研究所 http://www.jos.org.cn