Tuili(推理) 语言的编译方法与实现技术

高全泉

(中国科学院数学研究所)

THE IMPLEMENTATION TECHNIQUES OF TUILI

Gao Quanquan

(*Institute of Mathematics, Academia Sinica*)

ABSTRACT

Tools of universal interactive logical inference, Tuili in short, is a new style artificial intelligence language, it can do reasoning in different directions using various search strategies. It is very convenient to use Tuili for building expert systems and other knowledge-based systems quickly. In this paper, we will present the implementation techniques which are used in the implementation of Tuili.

摘 要

通用交互式逻辑推理语言Tuili是能够进行不同方向并能选择不同搜索策略进行推理的人工智能语言，特别适合快速建造专家系统或基于知识的系统。它的实现将使这一新颖的推理语言的应用成为可能。本文给出作者已经实现的一个Tuili系统中用到的编译方法和主要的实现技术。

§1. 引 言

逻辑程序设计在八十年代受到广泛的重视。于是，人们提出了种种对早期的逻辑程序设计的代表作Prolog 的改进方案。基本的思想是吸收别的语言的长处，弥补Prolog 本身的不足。这就是，在逻辑程序设计语言中增加或结合别的语言成分，或者反其意而用之。Tuili 就是一个以逻辑程序设计为基础，以建造专家系统和知识系统的实际需要为背

1989 年10 月29 日收到，1990 年1 月25 日定稿，中国科学院和国家自然科学基金资助课题。

景设计的知识推理语言。Tuili 的设计者是中国科学院数学研究所的陆汝钤教授。Tuili 代表“推理”，也是“Tools of universal interactive logical inference”的简写。关于Tuili 语言，见参考文献[1]、[2]。

Tuili 具有很强的表达能力和优雅的表达方式。逻辑程序设计创始人之一，R.A.Kowalski 在“Logic Programming”[3]一文中提出的元级推理、向前推理和向后推理的结合以及提高程序的计算能力等问题，在Tuili 中得到有效和谐的解决。Tuili 的设计富于创新。比如，语言中引进咨询谓词的概念，对大量应答式的交互式应用是既简单又实用；对表达式定义的扩展，使得模式匹配在更广的意义下进行，特别是串元表达式，能极大方便文字或字符串方面的应用；中断功能提供了改变程序正常执行的有效手段；允许使用外部语言，等等。对应用来说，语言的功能及其表达方式是问题的一个方面。另一个方面，一个语言的成功也应体现在能够在现有设备及技术条件下有效的实现。因此，Tuili 语言的实现无论对于证明它的可实现性，还是对于它的推广应用来说，都具有实际意义。

作者从1986 年开始着手实现Tuili 的工作。到1989 年9 月，实现了Tuili 语言的一个基本集(绝大多数功能)，即Tuili 1.1[4]。实现的主要功能有：向前推理，可以选择的搜索方式有广度优先、规则排序/ 循环优先、深度优先和最佳优先；向后推理，可以选择深度优先(以上推理包括元级推理)；以及中断、咨询功能和外部语言接口等。我们的实现工作表明，Tuili 可以在UV-68000 系列这样的微机上用C 语言实现。目前，Tuili 1.1 已经投入运行。应用Tuili 1.1 建造气象预报专家系统的工作正在进行。分布式逻辑推理系统D-Tuili 的演绎支持系统也建立在Tuili 1.1 系统之上。

§2. Tuili 1.1 的实现方法

一个Tuili 1.1 程序包括：图象基、数据基、过程基和规则基(数据基和过程基可以缺席)。我们先从观察以下两个简单的Tuili 1.1 程序开始，然后讨论实现Tuili 1.1 的一般方法。

例1: 语法分解(由底向上)

文法为1) S→ab;

2) S→asb;

任务为识别句子a^5b^5(归约到S)。

```
tuili (bottom-up);
 pattern_base;
  var 1: int;
      x:string;
      t1, t2:elem;
  pred sentence(x);
eof-pb;
database (db1);
    sentence ("aaaaabbbbb");
eof db1;
rulebase (rb1, 0);
  start: - >in(db1)^out(db1)^depth^ford^ goals(sentence("s"))^run(rb2);
```

```
  fail:   – >print("error")^stop;
  done:   – >print ("allright")^stop;
 eof rb1;
 rulebase(rb2, 1);
  sentence(t1/"asb"/t2)– >sentence(t1/"s"/t2);
  sentence(t1/"ab"/t2)– >sentence(t1/"s"/t2);
 eof rb2;
eof tuili;
```

例2: 梵塔问题求解

```
tuili(hanoi);
   pattern_base;
     var x, y, z, w: int;
     pred move(x, y, z, w);
          inform(x, y);
          hanoi(x);
   eof_pb;
rulebase(control, 0);
    start: – >depth^back^goals(hanoi(5))^run(rb1);
   done:  – >print("allright");
eof control;
rulebase(rb1, 1);
   move(x, "left", "centre", "right")– >hanoi(x);
                                    – >move(0, x, y, z);
   move(w-1, x, z, y)^inform(x, y)^move(w-1, z, y, x)– >move(w, x, y, z);
   print("move a disc from"/x/"to"/y)– >inform(x, y);
eof rb1;
eof tuili;
```

例1 是执行数据驱动的向前推理，从数据"aaaaabbbbb" 出发，到推出目标"S" 时为止，搜索策略是深度优先。例2 是执行目标驱动的向后推理，由目标hanoi(5) 出发，到move(o, x, y, z) 时止。例中，run 所在的规则即是元规则。上级规则基(级别高) 可以通过元规则来启动下级(级别低) 的规则基，下级规则基在完成上级规则基指定的任务后，仍返回到上级规则基执行。因此，Tuili 1.1 中执行的推理是多数据基、多规则基、多推理方式并能调用外部语言程序的复杂推理。

采用一般的编译方法，比如: 将向后推理做类似于Prolog 到Pascal 的过程的转换[5]，将向前推理编译为Rete 网[6] 的方法。最突出的问题可能是: 假定前推理在先，后推理在后，则后推理无法使用在前推理执行过程中产生的新数据，因为那种静态的把数据(事实) 编译成过程的方法难以解决由前推理执行引起的数据基中数据的动态变化问题。同时，由于对不同推理方向、不同搜索策略和使用不同数据基的规则基产生的目标程序各不相同，既需要有多个编译程序 "各编其码"，又有同一数据基和规则基的多次编译问题。特别是带前探的最佳优先，最佳规则的确定依赖于候选规则的试探性执行结果，在编译时无法确定规则执行的次序。

为了避免上述问题，我们认为，Tuili 1.1 的实现应考虑以下两条原则:

1) 编译产生的目标具有通用性，能为不同的推理方式使用;

2) 目标代码的执行效率较高。

原则1) 容易达到，只要将源程序编译成一种“中间”代码，并构造各种推理方式的推理机，各个推理机按自己的工作方式执行代码表示的语义。对2), 有较大的可塑性。一般说来，代码执行的效率与它自身的表示级别有关。如果编码科学，具备较多的可执行因素，则执行效率就可保证。

我们采用的实现方法基于上述原则。具体就是：用一个编译程序对所有Tuili 1.1 程序进行编译，产生C 语言形式的目标程序，目标程序的运行产生程序的内部表示结构，即规则树和目标树(子树), 它们在以推理机为核心的运行系统的支持下运行。

把一个Tuili 1.1 源程序转换为可执行程序经过以下两个步骤:

1) 将Tuili 1.1 源程序编译成C 语言代码;

2) C 代码程序经C 编译程序编译并与运行支持系统链接(运行支持系统是“.O”文件)。

§3. 程序的目标结构

在Tuili 1.1 中，由编译子系统tcompiler 将Tuili 1.1 源程序编译成C 语言形式的代码程序。tcompiler 是用C 语言写成的一个分为两次扫描的编译系统，采用递归子程序方法实现。编译产生数据化的代码程序，代码程序经执行生成程序的内部表示。这里，我们主要介绍目标代码及其生成的可在运行支持系统的支持下执行的目标结构。

3.1 用于构造目标树的数据结构

规则和数据是程序的主要组成部分。它们的内部表示是目标(goal) 树结构。所有目标树存放在一个名为“堆数据区”的一维空间。“堆数据区”的数据类型叫“堆类型”，它满足

1) 可以容纳规则和数据中所有客体的代码;

2) 能够区别不同客体的代码的含义。

Tuili 1.1 程序中的所有客体分为数十种。各种客体的代码表示具有统一的表示形式。一个客体的代码表示分为类型值和实体值两部分。例如，整数、实数、变量、谓词、咨询谓词、中断以及系统设备等客体的代码表示中，都用一个唯一的类型值表明它所代表的客体，即表明它属于哪一类。实体值就是客体的值表示。不论客体多么复杂，它的值表示总可以用整数、实数或指针加以表示。这样，源程序中的整数、实数、变量等简单量，只需一个“堆”元素便可构成其直接的内部表示。复杂量，如谓词、各种表达式等，其内部表示的“目标树”用一组堆元素构造。

堆类型的定义如下:

```
struct heapsp {   /* 堆数据的类型定义*/
    char type;    /* 类型域*/
    union {       /* 值域*/
    long int ival; /* 长整数*/
    float fval;   /* 浮点数*/
    int *ipt;     /* 整数指针*/
```

```
char *str;              /* 字符串指针*/
float *fpt;             /* 浮点数指针*/
PFI fpti;               /* 整函数指针*/
struct heapsp *php;     /* 间接指针*/
PFF fptf;               /* 实函数指针*/
struct presuxin *suen; /* 属性指针*/
}uval;                  /* 值域是一联合*/
};
```

这种数据结构也满足运行阶段表达式的计算结果的值的存放，因为它可以象无类型变量那样使用。以后我们将会看到，对于Tuili 1.1 中的重要成份算术表达式的“表达式树”的表示，这种数据结构的使用不仅是合理的，而且是经济的。

3.2 各个成份的目标结构

3.2.1 图象基

图象基的目标结构是：常量、变量、形式函数和谓词的定义表，统称原子表。以谓词定义表为例，该表的每个元素包括：谓词名、参数个数、参数的类型表、说明串1、说明串2 等。原子表的存在一是因为运行阶段需要引用其中的信息，二是为了将内部表示的程序成份还原为源程序的形式(在用户界面里用到)。

原子表的目标代码形式是C 语言相应类型的结构数组，这些数组用有序的原子信息的初值表初始化。

3.2.2 数据基

每个数据(即实在谓词) 的“目标树”结构我们在3.2.4 里会看到。这里，仅就目标形式(内部) 的数据基的组织进行讨论。

目标数据基的个数是系统数据基的个数(4) 与用户定义的数据基的个数之和m。每个目标数据基按谓词索引形式组织：为其分配ps(ps 是谓词表的长度，即谓词个数) 个索引单元，每个索引单元有头和尾两个指针。同名谓词的数据组织成“队”结构。名字为谓词表中第i 个元素的名字的数据的头和尾分别由目标数据基的第i 个索引单元所指。目标数据基的总体结构如图1 所示(图中i=0, 1, ···, m-1; j=0, 1, ···, ps-1)。

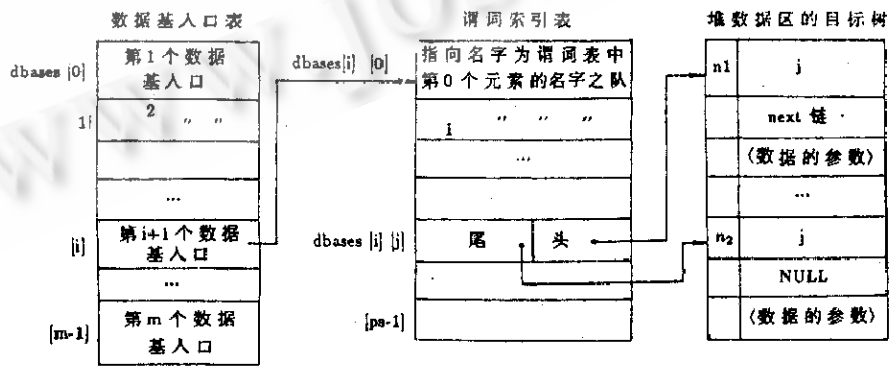


图 1

3.2.3 过程基

过程基的目标代码包括三部分

1) 原来过程基中的C 函数定义;

2) 接口程序。每个在过程基中定义的函数都有一个与之对应的接口程序;

3) 过程基中定义的函数的属性表。

Tuili 1.1 程序中，允许在goal 级和参数一级调用外部语言C 程序。这意味着：以Tuili 1.1 目标程序的运行环境调用C 语言函数。接口程序的作用是根据原函数的指针和运行时的实参表的指针，“组装”合乎C 语言文法及类型要求的函数调用，并返回调用的值。运行阶段，对所有外部函数的调用，都通过(< 接口程序指针>)(< 原函数指针>, < 实参表指针>) 这种简单统一的指针调用方式，获得外部函数的调用值。

例：设过程基中有以下函数定义

```
procbase(compute);
    sum1(j, k, 1)
    int j, k, 1;
    {
     if (k==0) return(1);
     else return (j*1);
    }
eof compute;
```

由编译程序产生的sum1 的接口程序如下:

```
int av0(a, b)
int (*a)( );
struct heapsp *b;
  {
int r;
int y0;
int y1;
int y2;
y0=((*(b+0)).type==4)?(*(b+0)).uval.ival: (*(b+0)).uval.fval;
y1=((*(b+1)).type==4)?(*(b+1)).uval.ival: (*(b+1)).uval.fval;
y2=((*(b+2)).type==4)?(*(b+2)).uval.ival: (*(b+2)).uval.fval;
r=(*a)(y0, y1, y2);
return(r);
}
```

3.2.4 规则基

所有规则基中的规则的内部表示是一片森林。每条规则是森林中的一棵二叉树，由左、右两个分枝组成，每个分枝上的m 个叉表示m 个目标(goal)。每个规则中的目标(包括数据基中的实在谓词) 又是一棵有n 个叉的目标树(goal 子树)…，它的一片叶子是常量、变量、直接量的直接的代码表示。规则基的目标结构除了规则索引表和规则本身的目标树以外，还有关于森林表示的静态信息表，我们先给出规则:

move(w-1, x, z, y)^inform(x, y)^move(w-1, z, y, x)→move(w, x, y, z); 的树型结构(图2)。

规则中的目标以及数据基中的实在谓词，如：谓词、函数调用、内部成分等顶级目标，目标结构是n 叉树形式(假定有n 个参数)。对普通谓词p, 其根表示是：类型值=70, 实体值=i; i 是p 在谓词表中的下标位置。n 个叉分别是n 个参数的子树或叶表示。再如，对

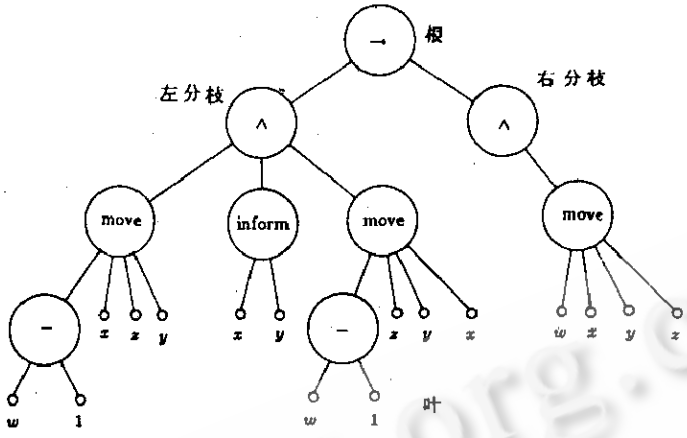
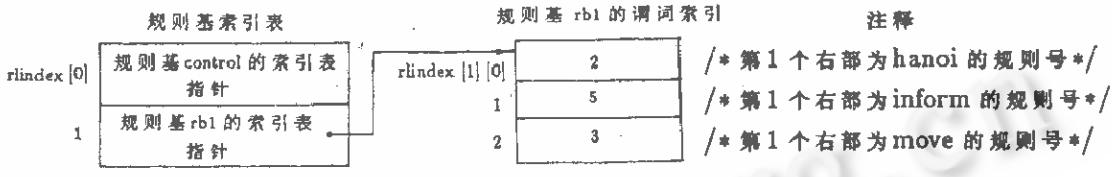


图2　规则的树型结构

于外部函数调用f 来说，根表示是：类型值=f 在函数表中的位置，实体值= 函数f 的指针，另有一个辅助单元，存放f 的接口程序的指针。参数位置上的各类表达式分别是二叉树或n 叉树。

现在我们给出 §2 中例2 的规则基的目标结构。为了简洁，只选取本节树型图对应的那条规则的目标树的具体代码。

1) 规则基索引表



2) 静态信息表(每个表元素有若干域)，如图3 所示。

§4. 运行支持系统

运行支持系统即是规则树的执行系统。其总体结构如图4 所示。

运行支持系统的核心是推理机a、b、c、d。系统总是首先启动推理机a，执行o 级规则基。若执行到元规则，便启动由该规则里的关于推理方向和搜索策略决定的推理机，执行由run 的参数给出的那个规则基，设为rbi。rbi 执行结束，运行rbi 的推理机即停止，控制返回到原来的推理机，继续执行位于run 所在规则之后的规则(不包括一些特殊情形)。

推理机执行的对象是规则基里各规则的规则树。同一规则树若处在不同的推理机中，则以不同的方式执行，即：先执行左分枝或先执行右分枝等。运行时，一个规则树做一个过程对待，为其动态分配变量空间。

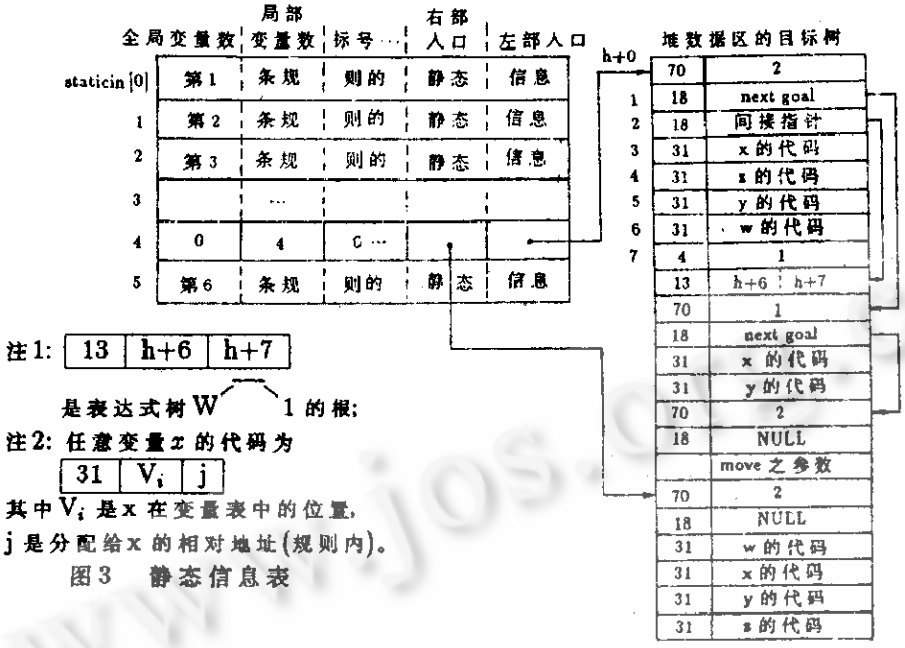


图3　静态信息表

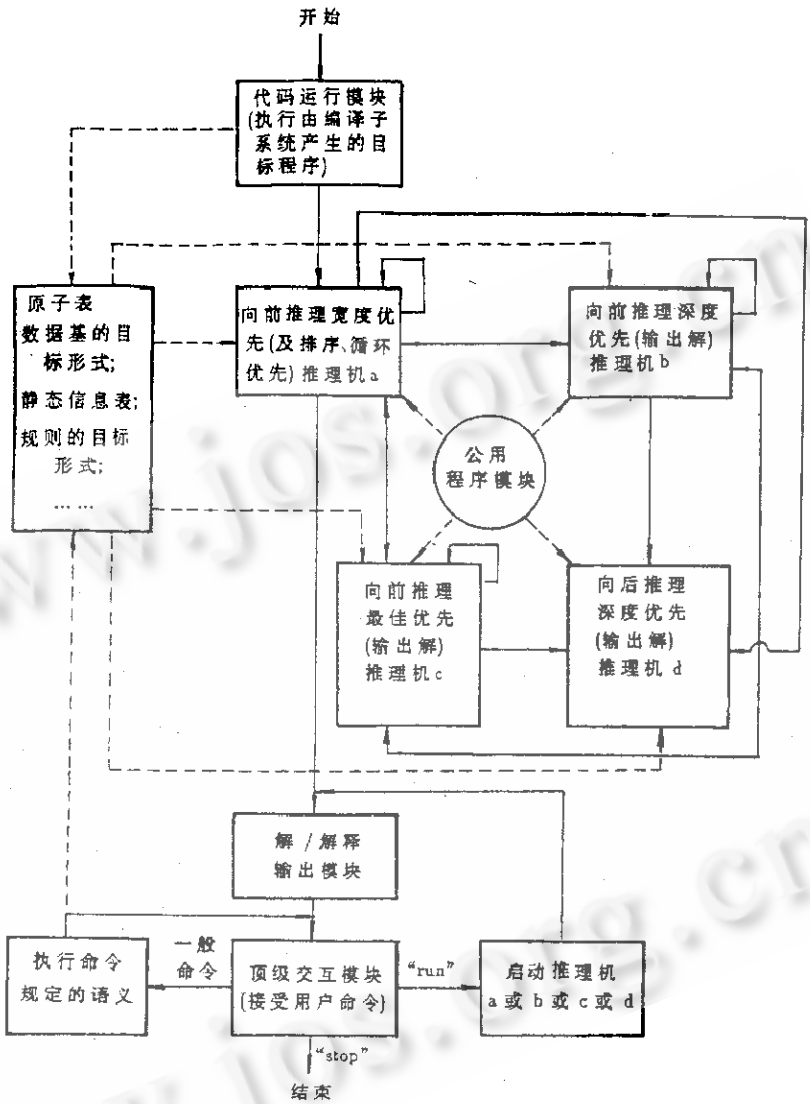
每个推理机都有自己的运行栈，各自以不同的方式执行规则树，并按自己的工作原理选取数据和规则。另一方面，不论哪个推理机，在执行一个具体的目标树时，采取的处理算法却是大同小异。因此，我们构造了一个公用程序模块，其核心是目标树的执行。于是，推理机不过是相应推理方式的基本控制算法的实现，它们调用公用程序模块完成每个具体的目标树的执行。关于Tuili 1.1 中的多推理机结构和控制算法，我们将另文讨论。以下，主要讨论与目标树执行有关的问题。

4.1 目标树的执行

根据目标树的根的类型值，可知当前目标是哪一类，分别进行相应的处理。设当前目标是普通谓词p, 其根的值是i。对任一输入数据基d, 此时的动作是把p 的目标树与dbases[d][i].head 所指的实在谓词的目标树进行匹配；对于向后推理，倘前者失败，则对当时的规则基r, 将p 的目标树与由规则索引表rlindex[r][i] 的值给出的那条规则的右部的目标树p′ 进行匹配。若当前目标是外部函数调用，首先调用系统子程序passage 执行各参数的子树上的运算，将计算结果传递到动态分配的实参区，然后，以(< 接口程序指针>)(< 原函数指针>, < 实参区指针>) 的方式获得调用值。

假定目标是咨询谓词，根的类型值是69 或71(对应于不登记和登记咨询结果), 实体值仍是i。首先，将p 的目标树与dbases[3][i].head(系统样品库) 所指的目标树匹配。若匹配失败，执行一个交互式过程inpred。inpred 从谓词定义表predicates[i] 处获得输出信息，并从用户终端获得输入信息，作为变量的约束值。当根类型是71 时，要在dbases[3][i].tail 的位置上产生当前目标树的副本。

元规则里的目标执行时(除run 外), 收集诸目标提供的控制信息。执行run(r) 时，调用由控制信息里的推理方向和搜索策略确定的那个推理机过程，r 以及其它控制信息作



注：虚线表示信息流，实线表示控制流

图4　运行系统总体结构

为调用时的输入参数。

在目标树的执行里，应用映射原理和指针特性，不仅简单，而且快速。基本上不需要查表和比较谓词名、函数名等操作。

4.2 匹配

匹配是求两个谓词的最广通代意义下的一致匹配(unification)。它是运行支持系统里调用频繁的基本功能。根据目标树的结构，不难看出，匹配只需对两个谓词的n 对参数的子树进行匹配。由于参数位置上的数据对象是包括算术表达式在内的各种表达式和基

本数据，因此，匹配过程是一个复杂的过程。在将一对子树匹配之前，先执行表达式子树上的计算，然后根据子树或叶子的当前类型值(包括取变量约束值)，分别执行不同类型的匹配。限于篇幅，我们只给出串元表达式匹配时采用的规则。

两个串表达式(串元表达式或字符串)S_1 和S_2 中，至少有一个串元表达式(带连接符"/")，分以下情形讨论：

a. 若S_1 是串元表达式，S_2 是串；

1. S_1 的各运算分量若非变量和字符串(包括定义为串的常量)，匹配失败；

2. 设S_1 中包括m 个字符串$t_1, t_2, \cdots, t_m, m > 0$，

1) 在S_2 中，自左向右找出第一个子串t_1，然后从t_1 右边起自左向右找第一个$t_2, \cdots$，如此继续，直至找出全部$t_i, 1 \le i \le m$。

2) S_2 中，每个t_i 与t_{i+1} 之间的部分与S_1 中t_i 和t_{i+1} 之间的变量名匹配；

3) 若S_2 中t_1 之前(t_m 之后) 的部分非空，而S_1 中t_1 之前(t_m 之后) 没有变量名，则匹配失败。否则，令非空部分与相应的变量名匹配。

4) 如果在2)、3) 的匹配过程中，使同一变量名与不同的子串匹配，则匹配失败，否则匹配成功。

b. S_1 和S_2 都是串元表达式

设S_1 和S_2 分别含有n 个运算量e_{1j} 和$e_{2j}(j = 1, 2, \cdots, n)$，则

1) 对任一i，若e_{1i} 和e_{2i} 可以匹配，则称e_{1i} 和e_{2i} 匹配；

2) 若对所有i，e_{1i} 和e_{2i} 顺序两两匹配(即存在φ，使$\varphi(e_{1i}) = \varphi(e_{2i}), 1 \le i \le n$)，则$S_1$ 和S_2 匹配成功。

4.3 其它实现技术

1) 产生新数据时，前推理使用"结构复制"，后推理使用"结构共享"。

2) 后推理中，使用了"尾调用优化"和"成功出口处回收"技术，用以控制运行栈的增长和回收局部变量空间。

3) 在用户界面里，有"反编译"能力，将内部表示的变量、数据还原成源形式。

§5. 结　语

应用本文介绍的编译方法和实现技术，我们完成了Tuili 的首次实现。由于目前uv-68000 系列机上尚无与Tuili 接近的其它语言，因此要把Tuili 1.1 与其它语言的效率做比较是困难的。试用及应用Tuili 1.1 的实际工作表明，由它产生的可执行程序有令人满意的效率，不论在时间还是空间方面，都可以满足一般的应用。

对实现Tuili 的完整集来说，仍有许多需要研究和探讨的问题，有待于做进一步的工作。

Tuili 实现工作得到陆汝钤教授的帮助和支持，在此致谢。

参考文献

[1] 陆汝钤，"关于Tuili 的设计"，《计算机软件开发方法、工具和环境》，西北大学出版社，西安，1985。

[2] Lu Ruqian, "Tuili---A Language for Expert Systems", Advances in Chinese Computer Sciences, I, 1988.
[3] Kowalski R.A., "Logic Programming", Proceding of IFIP, 1983.
[4] Gao Quanquan, "An Implemented System of Knowledge Inference Language Tuili 1.1", Research Report of Laboratory of Mangement Decision and Information System, Academia Sinica, MADIS YB90-0015, 1990.
[5] J.F.Nillson, "对基于定义域的prolog进行编译", 计算机科学, 1986, 2。
[6] Forgy C.L., "Rete: A Fast Algorithem for Many Pattern/Many Object Pattern Match Probelm", Artificial Intelligence, Vol. 19, 1982.
[7] Christopher J.H., 《Introduction to Logic Programming》, London, 1984.
[8] W.F. Cloksin & C.S.Mellish, 《Programming in Prolog》, springer-verlage Berlin Heidelberg, 1981.
[9] Brian W. Kernighan & Dennis M.Richie, 《The C Programming Language》, Prentice Hall Inc, 1978.