DOI:10.13657/j.cnki.gxkxyxb.1989.01.001

PICASSO——一个用于模糊决策系统的基于规则的程序设计语言*1

符华儿编译

（广西计算中心）

摘要： 本论文介绍一种新的基于规则的程序设计语言，其名为PICASSO，它为具体执行模糊决策支持系统（Fuzzy Decision SupportSystem）而设计。PICASSO已在美国休斯顿大学计算中心的AT&T公司的3B2OS小型计算机上部分运行。整个系统用Franz Lisp书写。

PICASSO具有以下特征。它使用正向链接（Forward chaining）推理手段。该语言有三种控制策略：面向宽度优先的系统控制模式，用户控制模式和元规则（meta-rule）*2控制模式。PICASSO支持匹配变量以及其他类型的变量，以便于使用以模式匹配为主的调用方式。在该语言中，不同规则之间的信息能够用信息传送和数据共享两种方法来进行通讯、交换。PICASSO的规则工作在可永久保存的知识库上，这个知识库必须预先用称为符号图形（S—diagram）的一种强有力的数据模式来定义。PICASSO的推理器(infere nce engine)实施知识库的一致性。知识库中可以保存模糊信息（fuzzy information），这些信息用区间模式的二值法来表示。区间模式允许分配一个概率给一个事实（断言），同时还要表示我们相信这种概率估计的程度，这一点对于模糊决策支持系统来说是重要的，因为这种系统常牵涉到不同程度的经验知识的判定规则。有了对某些事实（断言）的概率估计，又有专家对此概率的相信程度（凭经验），就可较全面地选择某种决策。对于区间模式来说，我们已经提供了一些特定的运算公式，用于对不确定性（uncertainty）知识进行自动推理。PICASSO支持基于规则的程序设计语言和函数型的程序设计语言两者的组合并支持多个基于规则的程序设计语言和函数型的程序设计语言两者的组合，并支持多个基于规则的程序之间的协同操作：可以容易地把PICASSO程序综合成象一般的LISP函数那样的LISP程序，而且PICASSO程序本身也可以任意地调用LISP函数。

※1. 原文是用英文写的。作者是Christoph F.Eick和Huaer Fu（符华儿）符华儿曾在美国休斯顿大学计算机科学系学习、研究、工作，具体负责PICASSO系统的总体设计和具体实现。博士Christoph F. Eick是符华儿的导师。

※2. 元规则——用来写编译规则的一种系统程序设计语言

ABSTRACT

The use of rule basedproduction system oriented programming paradigmg has gained a large popularity in the last 5 years, especially for developin computerized expert systems. This paper introduces a new rulebased programming language called PICASSO that has been tailored for the mplementation of fuzzy decision support systems. PICASSO can be characterized as follows: It uses a forward chaining inference approach. The language give its user a choice of three different control strategies: breadth first oriented, user controlled and meta-rule controlled. PICASSO facilitates the formulation of pattern directed invocation schemas by supporting match-variables as well as other sorts of variables. Information can be exchanged between rules using message passing as well as data sharing. PICASSO rules operate on permanent knowledge bases that have to be defined in advance by using a powerful data model called S-diagram. The PICASSO inference engine enforces the consistency of the knowledge base. The knowledge base may contain fuzzy information that is represented using a two-valued approach called interval model that allows to assign a probability to a predicate as well as to express our strength of belief in this estimation, which is important for fuzzy decision support systems in which frequently involves rules that have very different degrees of empirical conformation. Special operators for automating reasoning in uncertainty have been provided for the interval model. PICASSO supports the combination of rule based and functional programming as well as the coorporation between multiple rule based programs: PICASSO-programs can easily be integrated to LISP-programs as ordinary LISP-functions and PICASSO-programs themself can call arbitrary LISP-functions.

1. 引　言

在人类生活中，为了判断某个物体的所在位置，或重新构造发生在过去的事件，或为了达到某个目标而制定计划等等，迫使我们进行推测。人类进行复杂决策的能力是人类智能最迷人的本事之一，特别是当牵涉到模糊的和不确定的知识时。象著名的福尔摩斯这种推理的天才总是吸引着人们的兴趣的。

通常，进行决策会牵涉到多种知识源，从中，专家通过运用一系列的规则（这些规则由专家的一般及专门领域的知识和经验所构成），会抽象出不同的线索。对于从分析不同的线索中得到的证据，专家进行综合，最后便提出了决策。因此，把人类的决策过程计算机化，即把专家的基于规则的推理过程用程序设计语言表达出来是十分吸引人的。总之，基于规则的程序设计语言在近年来的研究课题中已获得极大的普及，特别是在专家系统领域中。在这种系统中，专家的知识用条件／动作对（condition /action pairs）的集合来表示：

if（条件集）　then（动作集）

本论文的焦点是讨论一个名为PICASSO的、基于规则的程序设计语言，它已经被设计来进行自动的人类决策。在详细地介绍这个语言之前，我们将讨论几个现有的、基于规则的程序设计语言的优缺点，它们用于决策支持系统。

主要的基于规则的程序设计语言可以划分为两组：

（1）采用正向链接／数据驱动控制策略（forward chaining／data driven control strategy）的语言

（2）采用反向链接／面向目标控制策略（backward chaining／goal oriented control strategy）的语言

第二组语言经常用于进行诊断的专家系统，它们不适用于决策支持系统，因为：

·反向链接要求存在顶层断言／顶层断言的集合，然后从这些断言推出证据。然而，对于大多数决策支持系统来说，其主要的目标预先还不能给定，而是必须通过分析当前的情况而进行推理，即决策牵涉到多重的、经常发生冲突的目标。

·因为它们的工作是面向目标的，这意味着它们不能改变数据库的内容，因此这些语言不允许其规则的右边结论部分有任意的行动：能够出现在反向链接的规则右边的唯一行动是这样一些命令，这些命令把证据赋给某些断言，但是右手边不能包含那些改变现实世界／知识库的命令。即，反向链接的手段对实际应用是有限制的，在推理过程中，它的状态不能改变。显然，这种假设对决策支持系统是有限制的，特别是，如果在方案中涉及到有多个动作者做为伙伴或对手时，要判断哪一个对于所有有策略特性的应用情况来说是可行的，对这种问题的限制性更大。

象前面所谈到的，这同样的争论也出现在PROLOG的面向目标的程序设计风格中。

因此，我们把讨论局限在第一组语言上，我们打算研究这组中的两个非常流行的语言：LOOPS（〔LOOPS 83〕）和OPS5（〔BFKM 85〕）。

在OPS5 的正向链接手段中，规则由任意多的条件／动作对所组成。它通过激发（trigger，即，如果规则左手边的所有条件都满足，我们说这条规则被激发）、执行规则（执行被激发的规则的右手边的动作集）而把新产生的断言插入到工作内存（这是OPS5称呼

知识库的术语），或从工作内存中删除断言，或代换工作内存中的断言。OPS5的控制策略是优先选择带有最近的时间标签的规则，即，与知识库中最新的被改变的断言相匹配的那条规则首先被从冲突集（这个冲突集存放在一个循环周期＊3内被激发的所有规则）中选出执行；换言之，OPS5多多少少类似深度优先的控制策略。此外，OPS5的规则可以包含匹配变量，它给予该语言进行一阶断言计算（first order predicate calculus）的表达能力。它使用一个有效的算法——网算法（Rete-algorithm）（〔FOR 82〕）来进行模式匹配，同时，用以防止同一条规则被相同的限定值所激发而重复执行。

然而，OPS5有以下的缺点，它使该语言仅仅能部分地适用于决策支持系统：

• 它不提供基于模糊信息的推理概念。

• OPS5不支持永久知识库的概念：OPS5的工作内存不被保存，当一个OPS5的程序终止时，即每当一个OPS5的程序重新运行时，不得不重新生成知识库。

• OPS5为工作内存提供了数据模式，然而，这种模式是粗糙的，且只有非常有限的表达能力：即，完全错误的和不一致的信息仍能被插入OPS5的工作内存，而且推理器不能检测出这种错误，因为仅仅有非常少的一致性约束条件可以用OPS5的数据模式表达出来。显然，这种缺陷使OPS5程序的开发复杂化，因为当错误发生时不能立即被检测出来。

• 从每一件事必须在运行时完成的意义上说，OPS5是一个高动态系统，这导致了相当低的系统执行能力。

• 对程序设计人员来说，围绕着OPS5的控制策略是非常难于编制程序的，因为有时其他的控制策略，比如宽度优先或者最佳优先对于解决某些实际应用问题会更合适和有效，从这点来看，OPS5的控制策略有很大的限制性。

• 与大多数其他的基于规则的程序设计语言相比，OPS5的规则含有许多控制信息；即，当使用OPS5时，目标级的知识和控制信息不能明显地分开，这使OPS5的程序难于理解和维护。

• 没有提供与该策略有关的明确的方法和语言概念来解决如何把OPS5程序综合成一个复杂的软件产品，以及如何把OPS5的程序设计风格（比如面向函数或面向目标的程序设计）结合起来。

• OPS5的唯一通讯方式是信息传送（通过工作内存来实现）。该语言不支持规则间数据共享的通讯方式（对于数据共享方式我们将在此论文的第4章进行详细讨论。）

• 至于Loops，它支持不同的程序设计风格（我们将只把讨论局限在Loops的基于规则的特性方面）。Loops的知识库可永久存在；此外，Loops的知识库可以存在于不同的文本中。Loops的推理器使用一个透明而简单的正向链接控制策略来一条条地运行属于当前的活动规则集（active rule set）＊4里的规则。为了执行更复杂的控制策略，在运行时，程序设计员也有某些选择的自由，来动态地改变规则集。Loops的规则集是有序的条件/动作对，在这些集合中，通常采用用户定义LISP函数的那种方法来表示动作。此外，Loops支持把规则集视为模块单元的思想；这样就易于与已经使用其他程序设计风格的执行模块合并起来。

然而，Loops存在以下缺点，这使它做为决策支持系统的执行语言的能力受到限制。

• Loops的规则模式语言是非常简单的。与OPS5对比，Loops不能提供匹配变量的概念，这就限制了它进行命题计算的表达能力。

• Loops不能为知识（库）提供明确而清楚的修改功能；即，Loops的推理器不能控制知识库的修改，这在控制和优化推理过程方面造成了严重的问题。此外，Loops既不为定义知识库提供数据模式，也不为实施知识库的一致性提供有关功能。

• 与OPS5相比，Loops的推理手段没有检测循环和冗余的规则的功能，因此，防止使用循环和冗余的规则是程序设计员的责任，这使程序设计复杂化——至少使规则集复杂化了。

• Loops不支持对不确定性知识的推理。

*3：一个循环周期——指推理器对活动规则集里的所有规则从头到尾运行一遍。

*4：活动规则集——指推理器正在其上进行推理活动的那个规则集。

2．PICASSO的主要目标

根据上面的分析，可以得出结论：现有的基于规则的语言仅仅能部分地适用于复杂的决策支持系统。在本论文的其他篇幅中，我们将报告一个研究课题PICASSO，它是一个基于规则的程序设计语言，致力于克服上述语言存在的一些弊病。开发PICASSO的主要目标是：

• 使面向问题解决方法的产生式规则（production rule）易于执行。

• 为复杂的以及模糊的决策支持系统开发一个通用的正向链接推理器：即，该推理器必须适用于对任何问题自动地进行探索性的搜索（检索）和计算机推理。

• 该语言支持基于规则的程序设计风格和函数的程序设计风格两者的组合，并且完全与LIPS兼容。规则、知识库和推理器就象传统的LIPS函数那样工作：根据某些输入参数，推理器将激发、执行规则，而且最终回送某些输出变量的值；此外，这样的基于规则的程序设计函数可以被编译，从而加快推理的速度。

• 为模糊知识的表示，以及为牵涉到的含糊、不确定和不完全性知识的自动推理提供强有力的功能。

• 用一个独立的、称为知识库管理系统(KBMS)的模块做为工具来具体执行 PICASSO以及维护知识库。

• 便利于联结任意的推理器：允许高层的推理器递归地调用较低层的推理器。

• 该语言是一种类型语言——知识库管理系统/PICASSO 的推理器将自动地进行类型检测。

• 通过匹配变量和其他变量来支持以模式匹配为主的调用方式。

在下面的章节中我们将对PICASSO进行详细的讨论，其中第3章讨论在PICASSO中如何定义和维护知识库。第4章讨论PICASSO规则的句法和语义。第5章讨论PICASSO所支持的控制策略。第6章介绍PICASSO在不确定性知识方面的推理手段。第7章介绍PICASSO不同版本的用户接口。第8章列举一个简例，说明PICASSO的工作步骤。第9章为参考资料。

3．PICASSO 知识库的管理

PICASSO的推理器对分配给它的知识库中存有的断言（事实）进行检索和修改，用这种方法进行规则的运算。本章我们将讨论PICASSO知识库所使用的数据模式，并描述允许PICASSO的规则用以读取知识库的内容及管理知识库的有关命令。

3.1 PICASSO的数据模式

我们用符号图形来描述数据库语义的概念数据模式。这里所说的数据模式已受到二元关系模式（〔ABR 74〕、SDM〔MCL 78〕），TAXIS——研究课题（〔MBW80〕，〔BOR 85〕）和在语义网上进行的人工智能研究的影响。论文〔EICK 87b〕中给出了这种数据模式的理论基础。运用这种数据模式来描述某些应用的实验情况已在论文〔Eick 84〕和〔Eick 85〕）中做了报告。现在我们开始设计符号图形来描述大型综合数据库的语义。本章，在进行简要介绍之后，我们将致力于把这种数据模式扩充到知识库系统。

在我们的方法中，数据库包含了对实体（entity）的描述。实体从属于类别（class）。实体的特征（property）用属性(attribute)来描述。一个属性有一个域类别（domain class）和一个范围类别（range class）。可以为一个域类别的成员分配零个，一个或多个范围类别的成员，而且每一个类别的成员可以做为一个属性值而出现零次，一次或多次。可通过使用标志unique（单值），multivalued（多值），onto（一一对应）和optional（任选）来限定属性的基数（cardinality）。也可以给属性指定省缺值。属性有两种，一种是描述整个类别的特征（比如，有一种属性，用以描述所有雇员的平均薪水）。另一种是描述一个类别的成员的特征（比如，每一个雇员的薪水）。数据模式会把它们区别开来。其中，第一种属性被称为类别属性；而第二种被称为成员属性，如无二义性，则简称为属性。

类别可以是其他类别的子类别（subclass），可把它们定义为与其他类别不相交（disjoint），也可定义为与其他类别重叠（overlap）。可为类别定义一些附加的限制，这些限制必须由所指定的类别的成员所遵从。

另外，还必须把以下的类别区分开来。一种是其成员已在计算机中被定义的（象整数，字符，一个记录类型的“地址”等）类别，它的范围不能随时间而改变（你不能把12从类别INTEGER（整数）中删去）。还有一种是不具备上述特征的类别（比如类别为汽车；显然，计算机预先未对汽车进行过定义，在我们的世界上，也没有一成不变的汽车：它会随时间而改变）。下面，我们把第一种类别称为词典类别（lexical class），而词典类别的成员将被称为值（value），词典类别的属性称为简单属性（simple attribute）。第二种是非词典类别（nonlexical class），其成员被称为实体(entity)，其属性称为作用属性（role attribute）。因为词典类别的范围不随时间而变，因此，为词典类别提供的唯一有用的运算是测试某个值是否属于（belongsto）一个指定的词典类别。对于非词典类别，其成员可被建立(create)、删除(delete)、可把它们连接到(connect)到其他类别，也可把它们从其他类别断开(disconnect)，而且，已赋予的实体属性值可由其他的实体值所取代(replace)。还可以测试一个实体是否属于（belongsto）一个给定的非词典类别。

列出了一个概念模式的子集，它反映了美国大学注册课程的主要内容。它由类别（class）course（课程）所组成。类别course包括了该大学课程表中提供的所有课程。course中提供section（节），因为学校可安排多门相同的课程，需用section把它们区分开。这里把section定义为另一个类别，它可做为类别course的作用属性（role attribuce），其成员成为这个作用属性的实体。作用属性section把课程的section与以下限制条件

联系起来——它具有多值（multivalued）或单值（unique）的特性，或可能根本没有值（optional），标志onto则表示一旦section有值，则必须与某个课程course一一对应（系统会自动地检测数据是否符合以上限制）。此外，对于每一个从属于某个course的section来说，有一些已被该section接收的学生，同时也有一些等待接收的学生。有时，一门course可能需要任意门必须预先具备的课程基础，因此，类别course的定义是递归的：即类别course的属性prerequisites（预先具备的知识）的数据类型是course（而不是象属性section numbr（节号）那样，其数据类型为Integer———整数）。

另外，类别Instructor（教员）和类别student（学生）是可重叠的，因为有些学生（研究生）也可能是教员（比如助教）。除非特别说明，要不然此数据模式的省缺假设是：类别是不重叠的（disjoint）。比如，类别Department（系别）和Instructor（教员）之间的关系未特别说明，故隐含地意味着它们是不重叠的。总之，该数据模式使用了不少省缺假设以便于对大多数的应用能进行最简短的描述。

我们的数据模式用于LISP环境。LISP的基本数据结构是符号表达式（S-EXPRESSION）。这意味着被存于知识库的每一种内容必须是符号表达式，要不然在LISP中不能处理。总之，词典类别S-EXPRESSION（符号表达式，连同这个类别的重要子集如ATOM（原子）或INTEGER（整数）在我们的数据模式中都被预先定义了。然而，用户仍有自由去定义进一步的词典类别，做为现存的词典类别的子集。比如，列（枚）举出词典类别的某些成员，或通过特定的模式（如定义数组、记录、队列或现存类别的成员集）来限定某个已定义的词典类别。

我们的注册方案中包括了以下用户定义的词典类别：COURSEID（课程号）、SEMMESTER（学期）、TIME-RANGE（时间范围）、GPA（到每一阶段为止的各课程平均分）和FACULTY（职员）。下面，我们给出其中两个类别的定义：

i）lexical class GPA subset of SEXPRESSION where（*a（and（numberp **）（greaterp ** 0.0）（greaterp 4.0 **））））

ii）lexical class FACULTY subset of SEXPRESSION where instances are（MATH COSC BIOL ECON GEOL PHIL）

其中，i）定义了GPA包含0.0——4.0之间的数，注意：如果要修改学生的GPA，知识库管理系统将防止把不合法的平均值（比如5.1或“abc”）分配给学生；而ii）通过列举出它的若干成员诸如MATH（数学）、COSC（计算机科学）、BIOL（生物）、ECON（经济）、GELO（地理）和PHIL（哲学）来定义词典类别FACULTY，上述这些成员都是词典类别SEXPRESSION（符号）表达式的子集。

总之，PICASSO的知识库是一个语义网络，它通过属性和类别的成员把某些实体与其他实体联结起来。在知识库中贮存的是对类别的成员进行描述的内容，即事实（fact）或一元断言（unary predicate）。

3.2　管理PICASSO知识库的命令

在这章的前面，我们介绍了PICASSO的数据模式。每个知识库必须与这种数据模式所说明的方式一致；即，如果对知识库内容的修改违反了上面所说明的概念模式，则PICAS

SO的运行系统将发布出错信息。PICASSO为知识库的管理提供了以下命令。

PICASSO共有35个命令，其中有用于知识库的，也有用于规则集的。比如命令infer（推理）、getcert（取确定因素）、setcert（置确定因素）和negcert（取反确定因素）是对断言的确定性因素进行处理和计算的；conclude类似于一般语言中的打印语句，等等。以下将陆续介绍几个命令，用户从中可略见一斑。

符号的约定：

〈·····〉 尖括号内的项是表示由用户提供的。

〔·····〕 方括号内的项是可选择的。

／ 斜杠表示可选两个选择项中的任何一个。

用命令create可生成一个新的实体，并把它存入知识库，其句法格式如下：（create 〈类别〉 〈相关表〉）。上述函数生成一个实体；该实体成为所说明的〈类别〉的成员（每一个实体必须至少是一个类别的成员）。〈相关表〉表示所指定的属性——值对（attribute-value pairs），这些属性——值对将分配给要生成的实体。最终，这个函数回送一个实体号（entity-number）做为结果，这个号码唯一地标明了在知识库中生成的实体。在实体的生存期间，此号码不会改变。在实际应用中，实体号常用于表示类别的成员，以及实体的属性：用一元断言（unary predicate）来贮存类别的成员（比如，（student 12）表示以下事实：由实体号12所标明的实体从属于类别student（学生），以及用二元断言（binary predicate）来表示属性（比如，（taught 13 45），其中tanght为属性，它表示由实体号13所标明的课程节号（section）是由实体号45所指定的教员来教（taught）的。或者，（st-ssn 12 43333333）说明：实体号12所标明的学生具有的社会保险号（属性st—ssn）是43333333。〈相关表〉可包括一个属性——值对，也可包括多个，它们构成了即将生成的实体的内容。比如：

例1：

（create section（（sectionnumber 1212）（semester Fall87）（offered-time（MO 200P 330P））（accepted-student 34）（accepted-student 33）（instructor 43））））

以上函数生成一个新的实体，它的类别是section（节），其中，相关表中包含了六个属性——值对，属性sectionnumbr（节号）的值是1212。属性semestor（学期）的值是Fall 87（87年秋季）；属性offered-time（提供的上课时间）的值是(MO 200P 330P)，意即：星期一下午2:00到3:00；属性accepted-student（已被接收的学生）的值是实体号分别为34和33的学生。

已经给定的属性——值对的内容可以用以下四个命令之一来修改：

amodify、areplace、ainsert和aremove。其中，上述每个命令的第一个字符a是assertion（断言）的缩写，它们的句法格式如下：

i）（amodify 〈实体号〉 〈相关表〉）

语义：为指定的实体的某些属性（由相关表指明）分配新值。

ii）（areplace 〈实体号〉 〈旧属性——值对〉 〈新属性——值对〉）

语义：用新属性值来取代所指定的实体中的旧属性值。

注：如果实体的某个属性含有多值，要改变其中的一个值，必须用areplace，而不是amodify。

iii）（ainsert〈实体号〉〈相关表〉）

语义：把由相关表所指明的属性——值对添加到指定的实体。

iv）（aremove〈实体号〉〈相关表〉）

语义：把由相关表所指明的属性——值对从有关的实体中删除掉。

下面我们给出一些例子：

例2：假定例1生成的实体的号码是111。

（amodify 111（（sectionnumber 4444）（instructor 76））

它意味着把实体号111中的属性sectionnumber的值从1212改为4444；把属性instructor的值从43改为76。

（ainsert 111（（accepted-student 56）（accepted-student 34））

它意味着把属性accepted-student的值56和34分别添加到实体111中。即，实体号为56和34的学生被接收了。

（areplace 111（accepted-student 56）（accepted-student 25））

它意味着用属性accepted－student的值25来取代原先的56。即，新接受的学生在实体号25，取代了原先被接收的学生（实体号为56）。

此外，用命令erase（擦除）可把指定的实体从知识库中删除掉。其句法格式是：

（erase〈实体号〉）

在执行完上述函数之后，有关的实体所牵涉到的所有属性和类别的成员都被删除掉了。

用命令infer可描述推理结果：（infer〈断言〉〔with〈区间概率〉〕）。

用命令connect（连接），可使已存在的实体成为新类别的成员。用命令disconnect（断开）也可从某个类别中解除对实体的连接。其句法格式如下：

（connect〈实体号〉〈类别〉〈相关表〉）

（disconnect〈实体号〉〈类别〉）

有时，你想检查知识库中是否包含着某个断言。在PICASSO，可用以下的任何一个方法来实现：

格式1：（〈类别〉〈实体号〉）

格式2：（〈属性名〉〈实体号〉〈实体号或值〉）

格式3：（not-exist〈断言／事实〉）

比如，（st-ssn 12 43333333）属于格式2，如果知识库中包含上述断言，则回送t（真），否则为f（假）。

对于命令not-exist，如果指定的断言不存在于知识库，则回送t，反之为f。

此外，在使用知识库之前，须用open把它打开，用完之后须用close来关闭以便系统把运行后的当前知识库存在磁盘上，若使用任选项cancel，则当前知识库不被存于磁盘。

4. PICASSO规则的详细说明

PICASSO规则的一般原则如下：

一条PICASSO规则由以下成分构成：

·规则名——为标识符，在整个系统中，规则名必须唯一。

·条件集——由“if”牵头，它们构成了一条规则的左手边（left-hand-side，简写为LHS）。

·动作集——由“then”牵头，它们构成了一条规则的右手边（right—hand—side，简写为RHS）。

·规则的附加特性——用于指示如何进行规则运算和推理的一些专用命令。

·规则的条件和动作部分由以下成分构成：

常数——任意的LISP原子（atom）和表（list）。

函数——用户定义的函数和LISP的内部函数，必须由%率头。

变量——局部变量（以？牵头）和全局变量（以&$牵头）。

PICASSO的规则用以下的句法格式表示：

i）（〈规则名〉（if〈条件集〉）（then〈动作集〉）〔〈附加的 特性〉〕）

ii）（〈规则名〉（perform〈动作集〉）〔〈附加的特性〉〕）

其中ii）是特殊情况，它表示条件集的结果永远为真，故直接执行动作集的内容即可。

例3：

```
(rule123 (if (salary ?z ?u) (%greaterp ?u 50000) (then(connect ?z well-paid-employee)))
```

此规则的含义是：所有薪水高于50000的人员属于高薪，把他们的信息联结到类别well-paid-employee（高薪雇员）中。

一条规则的动作部分（RHS）可做以下事情：

·它可以明确地把值赋给变量。

·它可以通过插入、修改、删除，或通过改变知识库的断言的确定性程度来明确地改变知识库。

·它可以对任意的LISP函数或用户定义函数进行计算，这本身可能改变变量的值和知识库的内容。

PICASSO规则的条件部分（LHS）可做以下事情：

·给变量赋值。

·检索存于知识库的断言和其确定性因素（certainty factor）。

·调用以回送一个确定性因素做为计算结果的、任意的LISP内部函数或用户定义函数。

PICASSO有两种不同的变量：

·局部变量（规则变量）

这种变量的生存时间限于所在的那条规则的活动期间；即，在完成了某规则的运算并跳出此规则之后，这条规则中所包括的所有局部变量也成为无定义的了。局部变量以？牵头，比如?X就是一个局部变量，它表示以下的语义：i）在规则的整个LHS成功地与知识库进行匹配之后，所匹配的那些值（单值或多值）被赋给与这对应的局部变量?X。

ii）显然，必须先成功地对局部变量?X进行匹配，让其获得限定值，然后才能被使用，要不然它本身没有值。由于这种变量的生存时间有限（一离开规则，它就失去原有的值），程

序设计员可以自由地在规则中使用这种变量（不同的规则可使用同名的局部变量，相互之间不会有任何影响）。对局部变量的使用没有任何限制。

现在我们解释一下例3的工作过程，看看局部变量的作用。

在例3中，规则rule123的LHS的第一个条件是（salary ?z ?u），其中?z和 ?u是局部变量，salary（薪水）是属性，?z表示实体号，?u表示属性值。一旦和知识库的内容匹配成功，?z和?u将分别含有实体号和薪水值。它们的关系是一一对应的，因为每个工作人员都有一份薪水。假如知识库中存有以下内容：

entity—number 实 体 号	name 姓 名	age 年 龄	title 职 称	salary 薪 水
3	John	42	高级工程师	56000
5	Kim	30	高级工程师	30000
6	David	66	高级工程师	60000

条件（salary ?z ?u）分别成功地与知识库中的下述内容相匹配：

（salary 3 56000）、（salary 5 30000）和（salary 6 60000）

于是?z和?u分别被赋予以下值：

变　量　名		?z	?u
值	①	3	56000
	②	5	30000
	③	6	60000

LHS的第二个条件是（%greaterp ?u 50000）其中，%greaterp是一个PICASSO函数，它直接调用LISP的内部函数greaterp进行两个数大小的比较。在上面的第①组值中，?u为56000，第③组值的 ?u为60000，都能使函数（%greaterp ?u 50000）和计算结果为t（真），这两组值被成功地保留，而第②组值中的 ?u为30000，使函数的结果为假，故去掉这组值。换言之，在对 rule123的整个LHS进行计算之后，找到的第①和第③组值是使LHS的所有条件同时满足的，即，这两组值激发了规则rule123。接着，PICASSO执行rule123的RHS。此时，RHS只有一个动作，就是：

（connect ?z well-paid-employee）从上面的匹配结果，?z的值分别为实体号3和6，而实体3和6分别对应的整个实体的内容是：

entity—number 实体号	name 姓名	age 年龄	title 职称	salary 薪水
3	John	42	高级工程师	56000
6	David	66	高级工程师	60000

于是这二个实体被连结（connect）到类别 well-paid-employee（高薪雇员）中去。

全局变量：这种变量的生存时间是整个推理过程。可把它视为标准的LISP全局变量，但其生存时间只限于PICASSO的运行期间。全局变量分别以&（赋值）和$（读值）牵头，比如&x和$x都是全局变量，它们的语义如下：

i）在把规则LHS的条件逐一与存在于知识库中的断言（事实）相匹配之后，如果匹配成功，则所匹配的对应值就赋给全局变量&x。&x可以有单值或多值。

ii）用$x读取已赋给&x的值。如果在一次成功的匹配之后，&x已被赋值（单值或多值），且其值会在不同的规则中被反复使用的情况下，使用全局变量是非常有用的。

例4：假定有以下规则：

（rule124（if（salary &x &y）（% greaterp $y 50000））（then（connect $x well-paid-employee））））

（rule125（if（age $x &z）（% greaterp $z 65）（name $x &n））（then（conclude The well-paid-employee whose name is $n and age is $z reaches retiring age））））

其中，rule124是例3中的规则rule123的翻版，只是把例3的规则rule123中的局部变量?z和?u分别用全局变量&x和$y来取代而已。假定知识库的内容不变，则在完成rule124的LHS的计算之后，赋给全局变量&x和&y的值（一一对应）分别为：

变量名	&x	&y
值	3	56000
	6	60000

这里，全局变量&x和&y起匹配获值的作用。在rule124中的全局变量$x和$y起读值的作用，此时$x和$y的值分别是前面赋给&x和&y的值，即：

变量名	$x	$y
值	3	56000
	6	60000

所以执行rule124后得有结果与rule123一样。但它们的区别是：推理器一旦跳离rule123，则rule123中的局部变量?z和?u的值便不再存在。但推理器一旦跳离rule124后，rule124的全

局变量$x和$y的值仍然存在，其后的任一规则都可引用它们。下面看看规则rule125：

rule125的LHS有三个条件，条件1为：（age $x &z），此时$x已含有值3和6（在rule124中从&x中得到的值），把$x的值代入条件1，条件1变成以下两个条件：

（age 3 &z）和（age 6 &z）

将上述两条件分别与知识库匹配，其结果为：

变量名	&z
值	42
	66

即

变量名	$z
值	42
	66

条件2为：（%greaterp $z 65），符合这个条件的$z的值是66，与之有关的$z为6，故上面$z为42与$x为3这组值被删去，只保存$z为66和$x为6这组值：

变量名	$z	$x
值	66	6

条件3为：（name $x &n）

把$x=6代入，条件3变成（name 6 &n），

匹配之后，符合这个条件的&n的值是David。所以满足rule125 LHS的三个条件的结果如下：

变量名	$x	$y	$z	$n
值	6	60000	66	David

于是执行rule125的RHS，RHS只有一个动作：（conclude The well-paid-employee whose name is $n and age is $z reaches retiring age）。

conclude是PICASSO命令，类似于其他语言的打印语句，其句法格式是：（conclude〈确定的事实〉）

所谓确定的事实是：常数、变量、函数表达式和字符串等等。一般用于在规则中写推理结论。比如rule125的conclude（结论）为：The well-paid-employee whose name is David and age is 66 reaches retiring age（名为David的高薪雇员，他的年龄是66，达到了退休年龄）。从中可见，全局变量$x的值被一个以上的规则所引用。

全局变量可用于保存中间结果，这些结果可供不同的规则使用。以往标准的正向链接手段总是把中间计算结果写回到知识库，它们必须这样做是因为缺乏保存这些数据的结构，这就导致在运行时必须常常重复地去知识库查找这些中间结果，从而严重地降低了工作效率。PICASSO提供了全局变量，就克服了这种弊病，而且使规则的计算参数化。通过这些变量，可以把推理过程的计算结果直接回送给调用者。显然，全局变量允许规则之间用数据共享的方式来进行通讯。因此，PICASSO为规则间的通讯提供了两种基本的方式：通过使

用全局变量的数据共享方式和通过修改知识库的信息传送方式，而大多数基于规则的程序设计语言只提供信息传送方式。

ii）此外，全局变量还可定义为输入变量，它允许我们使推理器的调用参数化。同时，也可把全局变量定义为输出变量，当推理过程结束后，它使推理器回送赋予变量的值。这就为我们处理有多重结构的推理过程提供了极大的灵活性。

5. PICASSO的控制策略

PICASSO的推理器工作于规则集，正工作着的规则集称为活动规则集。在推理过程中，由推理器的控制策略决定哪一条规则被激发执行。PICASSO支持三种不同的控制策略，用户可根据实际而选择使用。为了清楚地了解我们要介绍的策略手段，我们需要引入一些数据结构和概念来表示我们的算法。这里我们主要把侧重点放在说明算法做什么事情，而不是它们如何做这些事情。我们打算用尽可能简单的方式来表示它们，为此，算法将被表示为一系列的函数定义。而函数的定义采用以下格式：

Let 函数名（参数）= 函数体

函数体中可包含一系列的函数调用、命令、if语句、for语句或循环语言。如有一个以上的调用、命令或表达式，我们将用花括号｛·····｝把它们括起来。其中，if语句的格式是：

if 〈条件〉 then 〈函数体〉
else if 〈条件〉 then 〈函数体〉
·
·
·
else 〈函数体〉

〈条件〉可以为函数调用或布尔表达式。而for语句的格式是：

for each 〈变量〉 的 〈变量表〉
until 〈条件〉
〈函数体〉

这个语句描述了〈变量表〉中的一个变量或〈变量表〉中的所有变量（如果省缺until这部分的话），如果〈条件〉为假则执行〈函数体〉，为真则跳出这个语句。循环语句的格式是：

loop 〈函数体〉 until 〈条件〉

这个语句的〈函数体〉将被重复执行，直到〈条件〉为真。此外，我们用set命令来为变量赋值，它的格式如下：

set 〈变量〉 = 〈表达式〉

在〈函数体〉中，我们也可以用类似函数定义的方式来进行局部定义：

Let 〈名〉 = 〈函数体〉

这种定义的对象是那些用花括号括起的，包含着定义内容的函数体。我们的基本数据结构是表（list），用通常的表函数来进行处理，比如：null（x），head（x），tail

(x)，cons(x, y)，member(x, y)和append(x, y)。如果x为空，则函数null(x)的结果为真；函数head(x)传送一个非空表的第一个元素，而tail(x)包含从第二个元素开始的所有元素；函数cons(x, y)把元素x并到表y的前面而构成一个新表；如果x在表y中，则函数member(x, y)的值为真；函数append把x的内容添加到表y的后面而生成一个新表。

规则集由一系列规则组成。我们的规则有两部分：一系列前提(条件)和一系列结论(动作)。假设有规则r，我们可用antecents(r)和consequents(r)分别表示它们的前提部分和结论部分。知识库则由一系列的facts(事实)组成。

运用以上的数据结构和定义，下面我们就可以表示我们的控制算法了。

5.1 系统控制模式

系统控制模式的工作过程可简单解释如下：对于给定的一系列规则，系统控制模式的推理器要自动地推导出所有可能的结论。开始，它检查给定的规则集，如果发现该规则集是空的(没有任何供考虑规则)，它便带着已经找到的结论而退出整个推理过程，这些结论是一个facts(事实)表，该表中包括了知识库中的原始数据，也包括了推理过程中添加或修改的数据。如果规则集不是空集(即，还有许多规则供考虑)，它便取第一条规则，如果第一条不能被激发(not triggered，即这条规则的LHS不满足条件)，则依次检查后面的规则；如果这条规则被激发(triggered)，推理器便执行(fire)此规则的RHS，所产生的新事实被收在表new中。如果表new为空(没有新事实)，则系统控制推理器又去依次检查别的规则；如果表new不空(存在某些新事实)，则把它们存入知识库，然后，系统控制推理器又在新知识库的基础上，从头开始运行所有的规则。我们把此工作过程用前面介绍的函数较为准确地描述如下：

```
Let ( forward ( ruleset, facts ) =                                    (5.1)
   {if null ( ruleset ) then facts
    else
     {let r = head ( ruleset )
        if not doestrigger ( r )
        then forward ( tail ( ruleset ), facts )
        else
                    {let  new = canuse ( r )
                    if  null ( new )
                    then forward ( tail ( ruleset ), facts )
                    else forward ( rules, append ( new, facts ) )
                        }
                  }
      }
```

其中函数forward模拟了系统控制模式的正向推理器的工作过程，它有两个参数：ruleset为活动规则集的名字，facts是知识库的名字。此外，还用到了一些辅助函数。函数

oestrigger(r) (5·2) 测试规则r是否能被激发。仅当规则r的所有前提(antecedents)条件都能在知识库找到时(如果有命令not—exist或进行函数计算的条件，这些条件也须满足)，此规则才能说是被激发了。函数doestrigger及listand可表达这种功能：

```
let doestrigger(r) = listand(antecedents(r))                    (5.2)
let listand(s) = {if null(s) then true
    else if member(head(s), facts)
    then listand(tail(s)) else false}
```

函数canuse(r) (5.3) 累积所有由规则r产生的新事实(facts)，我们可以用另一个带有累积参数的函数usesome(cs, u)来做累积工作。如果所得的事实表cs中的各个事实x仍未存在于知识库facts中，则表明x是个新事实，它将添加到表u中：

```
let canuse(r) = usesome(consequents(r), nil)                    (5.3)
let usesome(cs, u) = {if null(cs) then u
        else
         {let  x = head(cs)
        if member(x, facts)
          then usesome(tail(cs), u)
          else usesome(tail(cs), cons(x, u))} }
```

下面我们举一个例子，说明系统控制模式的工作过程。

例5：假设有以下规则集A，它含有规则r1，r2，r3和r4：

```
((r1(if(e))(then(infer(f))))
 (r2(if(d))(then(infer(e))))
 (r3(if(c))(then(infer(d))))
 (r4(if(t))(then(infer(u)))))
```

设知识库KB中含有以下事实：

```
((C))
```

注：假设上面的c、d、e、t和u分别代表一些事实(断言)。

启动系统控制模式推理器后，它从头开始一条条地逐一检查规则集A中的每条规则，直到规则r3，它的LHS是条件(c)，与KB中已存在的事实(c)相匹配，LHS满足条件，即r3被激发，接着推理器执行r3的RHS，推出新事实(d)，它被添加KB，此时KB含有以下事实：

```
(    (d)    (c)    )
```

于是推理器又从头开始运行规则A，这次是r2被激发、执行，并推出新事实(e)，它又被插入KB：((e) (d) (c))。然后，推理器又从头开始运行A，这次是r1被激发、执行，并推出新事实(f)：

```
((f)    (e)    (d)    (c))
```

类似地，推理器又从头开始运行A，这次r1的LHS虽满足条件，仍被激发，但推理得到的事实(f)已存在于KB，它不是新的事实，不予考虑，于是推理器继续检查r2，r3，分别得到的事实(e)和(d)都不是新的事实，推理器便检查r4，其LHS不满足条件，此时规则A中的所有规则已检查完毕，ruleset(规则集)成空集，故整个推理过程终止，知识库KB中

的事实（包括添加的和原先存在的事实）：（（f）（e）（d）（c））就是结论。

这种控制策略有以下的优缺点：

·这种策略与宽度优先的问题求解策略相对应，可以对它进行以下模拟：知识库包含了检索、推理问题的一些事实或状态。推理器或者在知识库的未扩充的事实或状态上对规则进行运算（推理）并把其结果添加到知识库，或者检测某个事实、状态是否是目标状态。在后种情况下，一旦找到了目标状态，则推理器便终止工作。此外，当推理器推导的事实在规则的某个循环周期中都未发生变化时（因为检索、推理问题的所有事实、状态都已遍历而仍未找到目标状态，即，此检索、推理问题无解），推理过程也会被中止。

·一般来说，用户不需要考虑规则的次序：当运用这种策略时，假定规则是完全独立的（彼此间无关）。

另外，推理器可用优化技术来为规则排序、

比如，（r11（if（A））（then（infer（B））））

应该在（r12）if（B））（then（infer（C））））

之前使用，这样就可节省一次规则的循环周期。因为很多优化工作依赖于专门的、适用于特定领域的知识，所以这类优化工作不能兼并到一个通用的优化算法中去，仅仅有少数诸如此类的优化能自动地被检测、执行。

·由于知识库总在变化，在这种动态的变化环境中自动地终止推理过程还存在问题。

·正向链接推理器的另一个问题是，在不同的循环周期中，防止用相同的事实（限定值）来重复地激发、执行同一条规则。我们有解决这个问题的算法，然而，阻止运用冗余规则的算法所付出的附加费用是较大的。

·宽度优先正向链接策略——以系统控制模式为基础——对某些特定类型的问题来说效率不高。

5.2 用户控制模式

在用户控制模式中，它的正向链接推理器是按规则在活动规则集里的顺序逐一地从头到尾对各条规则进行判断推理的。如果活动规则集里的最后一条规则已被运行了，我们说规则集的一个计算周期已经完成。如果在这周期里知识库的内容改变了，则推理器自动地从第一条规则开始，进行另一个循环周期的运算；如果知识库的内容不变，则推理器自动地终止推理过程。但是，程序设计员可以按需在规则的RHS用控制转移命令把控制权进行动态转移。程序设计员能做以下事情：

·他可以用（continue　〈规则名〉　〈规则集〉）把控制权转到指定的规则集里的有关规则中。

·他可以用（restart〈规则集〉）使推理器从第一条规则开始进行另一轮的运行。

·他可以用命令insert—rules，delete—rules和replace—rules来添加、删除或取代活动规则集里的规则（一条或多条），也可以改变活动规则集的规则顺序：

i）（insert—rules〈规则名1〉from〈规则集1〉to〈规则集2〉first / last / prior〈规则名2〉）

语义：把规则集1中的规则1插入到规则2（在规则集2中）的指定位置。

ii）（replace-rules〈规则名1〉in〈规则集1〉by〈规则名2〉in〈规则集2〉）

语义：用规则2（在规则集2中）来取代规则1（在规则集1中）。

iii）（delete—rules〈规则名〉from〈规则集〉）

语义：把指定的规则从有关的规则集里删掉。

·他可以用命令stop或exit来中止推理器的工作。

以下我们举个例子说明之。

例6：将例5中的规则顺序稍加改动，并加一条规则r5，使之变成规则集B：

```
((r3(if(c))(then(infer(d))))
 (r2(if(d))(then(infer(e))))
 (r1(if(e))(then(infer(f))))
 (r4(if(t))(then(infer(u))))
 (r5(perfom(stop))))
```

假定知识库KB中仍象例5那样只保存有事实（c）：　　（（c））

启动用户控制模式的正向链接推理器后，推理器按序从头到尾逐一检查、判断活动规则集B中的各条规则：

r3：由于其LHS的条件（c）得到满足，推导出新事实（d），把它存入KB，KB现在内容为：　　（（d）（c））。

r2：由于其LHS的条件（d）得到满足，推导出新事实（e），把它存入KB，KB现在内容为：　　（（e）（d）（c））

r1：由于其LHS的条件（e）得到满足，推导出新事实（f），把它存入KB，KB现在内容为：　　（（f）（e）（d）（c））

r4：LHS条件不满足。

r5：命令perform在PICASSO中是“if—fhen”形式的另一种特例，它表示if部分的结果总是为真，不必加以判断，直接执行then部分（动作部分）即可。这里，动作部分是命令stop（停止），于是整个推理过程结束。KB中保存的最后内容为（（f）（e）（d）（c）），与例5的一样，但这里只对规则集B进行一次（一个周期）运算，而例5对规则集A进行了比例6多得多的循环计算。

例6是在r5中用命令stop终止推理过程的。stop可根据需要出现在任何位置。如果把例6中的r5这条规则去掉，会出现什么情况呢？在运行之前，知识库内容为（（c）），在第一个周期的计算中，当推理器运行到r4之后，第一个周期的计算结束，此时KB的内容为：

（（f）（e）（d）（c））

现在知识库的内容比原先的多了新事实（f）、（e）和（d）。于是，推理器自动从第一条规则r3开始，进行第二个周期的计算，同前面一样，r3、r2和r1分别推出以下事实：（d）（e）和（f），当推理器运行完最后一条规则r4后，它对在这一个周期的计算中所产生的事实进行检查，发现它们全都不是新事实（它们全都在第一个周期中已生成的旧事实），故推理器自动终止推理过程。即，在例6中，如果没有规则r5，则推理器要多运行一个循环周期才能自动终止。

用户定义模式有以下优缺点：

• 容易执行任意的、面向实际应用的控制方式。

• 在一个循环期内，可以有一条以上的规则被激发、执行，也可能没有。这对于递归规则的使用是有利的。此如在例6中，形成一条递归推理链：

$$(c) \xrightarrow{r3} (d) \xrightarrow{r2} (e) \xrightarrow{r1} (f)$$

• 由于仅仅激发、执行那些与专门问题有关的规则，所以容易兼并专用领域的控制知识，这有可能使推理过程具有好的性能。与系统控制模式比，这种模式的执行顺序与规则的顺序有关，程序设计员必须为安排规则的顺序而费心。

• 由于规则中包含控制信息，虽有极大灵活性，但维护规则集较困难；即，控制知识和目标知识未清楚地分开，这使规则之间有联系从而使软件设计过程复杂化。

5.3　元规则（meta—rule）控制模式

元规则是用元规则语言来书写的一种规则。元规则语言基于以下基础：目标规则（即，前面常说规则集里可被激发、执行的规则）是一种组合结构，它的元素是可以存取的；元规则可以检查目标规则的例示（instantiation）的内容。什么是规则的例示呢？如果一条规则的LHS匹配成功，则LHS（包括RHS）所有变量都用其匹配值来取代后的内容称为该规则的例示。规则的例示是一个有序表，该表的第一项是规则名；第二项是事实(断言)表：它依次保存与规则LHS的每一个条件相匹配的事实；第三项是该规则RHS(其中涉及到的变量都已用其相应的值取代）的全部内容。元规则是这样的一种规则：它指明按怎么样的方式来使用目标级的规则。它标明目标级规则的实用性或描述两个规则例示集之间的关系。在例7中，我们用中文和LISP语言两种形式来举个有关元规则的例子。第一条元规则指出：有关石油成品的哪些规则不太可能被应用。第二条元规则指出：建议进行短期投资的规则应该比建议进行长期投资的规则先得到应用。

例7：

metarule001

如果有规则说到石油成品，那么所有这些规则中的每一条都不太可能（0.7）被使用。

```
(metarule001
    (IF (RULES (MENTION IF-PART oil) LIST0))
    (THEN (CONCLUDE LISTO UTILITY N0 0.7)))
```

Metarule 002

如果〔1〕有规则谈到短期投资

〔2〕有规则谈到长期投资

那么，非常有可能（0.8）在使用与〔2〕有关的规则之前使用与〔1〕有关的规则

```
(metarule 002
  (IF (RULES (MENTION EITHER SHORT-TERM) LIST1)
      (RULES (MENTION EITHER LONG-TERM) LIST2))
  (THEN (CONCLUDE LIST1 BEFORE LIST2  0.8))
```

用元规则（meta-rule），我们可以推导出关于目的规则的例示的实用性程度或它们

之间的顺序关系的信息，然后，在每一个推理周期中选择一个最有希望（最好）的规则例示来加以执行。实用性（utility）程度的范围从－1到1，比如，如果规则的例示集绝对没有用，则实用性＝－1，其他的可能性在－1——＋1之间。

PICASSO的元规则控制模式的推理器主要采用三个主要步骤：

匹配——选择——执行

1）匹配

对于规则集里的每一条规则，元规则控制模式的推理器从头到尾依次进行检查、判断。如果某条规则被激发，则该规则的例示被存入到一个冲突集（cenflict set）中。前面说过，规则的例示是一个有序表，该表的第一项是规则名；第二项是事实表：它依次保存与此规则的RHS的每一个条件相匹配的事实；第三项是该规则RHS的全部内容。

这一步只是进行匹配并把规则集里所有被激发的那些规则的例示统统存在冲突集里，而不分别执行它们的RHS。

在匹配过程中，用的是网匹配算法（Rete match algorithm）来贮存匹配的结果。除第一个计算周期之外，从第二个计算周期开始，只将规则与知识库中被改变了的事实进行匹配，而不是重复地与知识库中的所有事实进行匹配。即，每当规则生成、修改或删除知识库中的某个（些）事实时，本算法所要做的事是标明由于知识库中某个（些）事实的改变而产生的那些不再被激发的规则（原先是被激发的），和那些新近被激发的规则（原先是不被激发的），同时修改与此有关的冲突集里的内容，从中删掉不再存在的例示。

2.选择

当一个计算周期结束时，选择器从冲突集里选择一个最好的例示。选择算法如下：选择器按照嵌在本推理器中的元规则所定义的实用性程度，对冲突集里的所有例示进行实用性排序，排在最前面的例示是实用性最大的。

3.执行

执行由选择器选出的例示的RHS，对知识库的内容进行修改。

以上三个步骤重复进行，直到在某个新的一轮计算周期中，知识库的内容不再被修改，则推理过程自动终止。

下面我们用比较精确的函数语言来表达元规则控制模式的工作过程。首先我们要用到两个数据结构：New－Facts和Halt－Flag。New－Facts是一个事实表。在这个表中，每个事实是一个前面带＋（添加）或－（删除）号的符号表达式。Halt－Flag是个终止标志。此外我们还有以下的函数：

i）METAINF（　　）

把元规则运用到新的例示去标明它们的实用性程度或建立它们之间的关系。

ii）SELECT－BEST（　　）；

从conflict－set（冲突集）中选择一个最有希望的例示

iii）FIRE－RULE（inst）

执行例示(instantiation,简写为inst)的then部分。这个函数回送一个事实表，这些事实可能被添加到知识库,或从知识库中被删掉或修改。以下是元规则控制模式的推理过程：

let inference（rules,facts）

```
{ set conflict-set = nil
set Rule-Name-Set = RULE-COMPILE ( rules )
set New-Facts= facts
set Halt-Flag = nil
loop
{ STORE-FACTS ( New-Facts,Rule-Name-Set )
METAINF (      )
set inst = SELECT-BEST (      )
 if inst = nil then
    set Halt-Flag= true
else
    set New-Facts= FIRE-RULE ( inst )
}
until Halt-Flag = true
}
```

其中函数inference模拟了元规则控制模式的正向推理器的工作过程，它有两个参数：rules是活动规则集的名字，facts是知识库的名字。

下面我们用一个例子来说明元规则控制模式的简要工作过程、对于语言的细节就不一一介绍了。

例8：设有以下规则集：

```
( ( RULE 001 ( if ( R  ?x ) ( F  ?y  ?x ) ( %greaterp ?y  60 ) )
              ( then ( create ( R ?y ) )
                     ( create ( A ?y ?x ) ) )
              ( priority 10 ) )
  ( RULE 002 ( if ( R ?x ) ( M ?y ?x ) )
              ( then ( create ( R ?y ) )
                     ( create ( A ?y ?x ) ) )
             priority 7 ) )
  ( RULE 003 ( if  ( R ?x ) )
              ( then ( delete ( R ?x ) ) )
              ( priority 7 ) ) )
```

设知识库含有以下事实：

```
( ( R  30 )
  ( M  70  30 )
  ( F  65  30 )
  ( F  45  70 )
  ( M  80  45 )
  ( F  75  65 ) )
```

设元规则库含有以下元规则：

```
(META001
   (IF (RULES (MENTION THEN-PART create) list1)
       (RULES (MENTION THEN-PART delete) list2))
   (THEN (CONCLUDE list1 BEFORE tist2 1.0)))
  (META002
   (IF (RULES (GREATER priority 8) list3))
   (THEN (CONCLUDE list3 UTILITY YES 0.8)))
  (META003
   (IF (RULES (MENTION IF-PART 70) list4))
   (THEN (CONCLUDE list4 UTILITY NO 1.0)))
```

当启动了元规则控制模式的推理器后，便开始工作：

i）对规则集里的规则进行编译，产生以下网络：

```
RULE001
   if-part-->((R ?x)(F ?y ?x))
   then-part-->((create(R ?y)
                 (create(A ?y ?x))))
   const-->((greaterp ?y 60))
   right1-->nil
   right2-->nil
   left1-->nil
   left2-->nil
   priority 10
RULE002
   if-part-->((R ?x)(M ?y ?x))
   then-part-->((create(R ?y))
                (create(A ?y ?x)))
   const-->nil
   right1-->nil
   right2-->nil
   left1-->nil
   left2-->nil
   priority-->7
RULE003
   if-part-->((R ?x))
   then-part-->((delete(R ?y)))
   const-->nil
   right1-->nil
```

```
    left1—→nil
    priority—→7
```

编译之后Rule－Name－Set(规则名集)的内容为 (RULE001 RULE002 RULE003)

ii)将每个规则网if-part所描述的条件与知识库的内容进行匹配，匹配后各网络 变为：

RULE001

```
    if－part—→( (R ?x) (F ?y ?x) )
    then－part—→( (create (R ?y) )
                  (create (A ?y ?x)))
    const—→(greaterp ?y 60) )
    right1—→( (R 30) )
    right2—→( (F 65 30) (F 45 70) (F 75 65) )
    left1—→( ( (R 30) ) )
    left2—→( ( (R 30) (F 65 30) ) )
    priority—→10
```

RULE002

```
    if－part—→( (R ?x) (M ?y ?x) )
    then－part—→( (create (R ?y) )
                  (create (A ?y ?x) ) )
    const—→nil
    right1—→( (R 30) )
    right2—→( (M 70 30) (M 80 45) )
    left1—→( ( (R 30) ) )
    left2—→( ( (R 30) (A 70 30) ) )
    priority—→7
```

RULE003

```
    if－part—→(R ?x) )
    then－part—→( (delete (R ?x) ) )
    const—→nil
    right1—→( (R 30) )
    left1—→( ( (R 30) ) )
    priority—→7
```

在第一个计算周期中，以上三条规则if－part的条件全都分别匹配成功，即这三条规则全都被激发，于是到第iii)步。

iii)把所有被激发的规则的例示存入冲突集(conflict－set)，冲突集为：

```
(RULE001 ( (R 30 (F 65 30) )
           ( (create (R 65) )
             (create (A 65 30) ) )
(RULE002 ( (R 30) (M 70 30) )
```

```
                ( (create (R 70) )
                  (create (A 70 30) ) ) )
        (RULE003 ( (R 30) )
                ( (delete (R 30) ) ) ) )
```

iv) 用meta-rule (元规则) 对conflicl-set中的规则例示进行实用性排序。

由于元规则META003中指出：在规则的if-part谈到70的那些规则，其实用性为-1：UTILITY NO 1.0，而在冲突集里，规则RULE002的if-part谈到了70——(M 70 30)，所以RULE002的实用性为-1，因此，系统把RULE002从conflict-set 中删掉。这种删除也传递到网络中去。根据META001和META002，RULE001在conflict-set中排序第一。

v) 从conflict-set中选择、执行实用性最大的例示，这里选择、执行了RULE001，执行后把产生的新事实存入New-Facts：

```
( ( + R 65) ( + A 65 30) )
```

vi) 把New-Facts中的每个事实与前面存在的每个网络进行匹配，所得结果如下：

```
RULE001
    if-part——→( (R ?x) (F ?y ?x) )
    then-part——→( (create (R ?Y) )
                  (create (A ?y ?x) ) )
    const——→(greaterp ?y 60)
    right1——→( (R 30) (R 65) )
    right2——→( (F 65 30) (F 45 70) (F 75 65) )
    left1——→( ( (R 30) ) ( (R 65) ) )
    left2——→( ( (R 30) (F 65 30) )
              ( (R 65) (F 75 65) ) )
    pirority——→10
RULE002
    if-part——→(R ?x) (M ?y ?x) )
    then-part——→( (create (R ?y) )
                  (create (A ?y ?x) ) )
    const——→nil
    right1——→( (R 30) (R 65) )
    right2——→( (M 80 45) )
    left1——→( ( (R 30) ) ( (R 65) ) )
    left2——→nil
    priority——→7
RULE003
    if-part——→(R ?x) )
    then-part——→( (delete (R ?x) ) )
    const——→nil
```

```
right1—→( (R 30) (R 65) )
priority—→7
```

其中，RULE002未被激发，RULE001被一组值激发，RULE003被二组值激发，于是Conflict-Set为：

```
( (RULE001 ( (R 65) (F 75 65) )
           ( (create (R 75) )
             (create (A 75 65) ) )
(RULE003 ( (R 30) )
         (delete (R 30) ) ) )
(RULE003 ( (R 65) )
         ( (delete (R 65) ) ) ) )
```

运用元规则，第一个例示RULE001被选择。在执行这条规则之后，New-Facts为：(+R 75) (+A 75 65))，把这些新事实分别与各个规则网络中的if-part进行匹配，匹配成功的事实就添加到各相应的网络中去。各网络的内容变为：

RULE001

```
if-part—→( (R? x) (F ?y ?x) )
then-part—→( (create (R ?y) )
             (create (A ?y ?x) ) )
const—→(greaterp ?y 60)
right1—→( (R 30) (R 65) (R 75) )
right—→( (F 65 30) (F 45 70) (F 75 65) )
left1—→( ( (R 30) ) ( (R 30) ) ( (R 75) ) )
lefr2—→( ( (R 30) (F 65 30) )
         ( (R 65) (F 75 65) ) )
Priority—→10
```

其中，RULE001的两条例示在前面已先后执行过，不再予以考虑；RULE002未被激发，不予以考虑；RULE003分别被三组值激发，运用元规则META002，RULE003的三条例示均具有相同的实用性（UTILITY=0.8），可按任意的次序先后执行它们。这里，执行RULE003的任何一个例示都不会产生新的事实，New-Facts为空，系统自动把Halt-Flag置为真，整个推理过程结束。

由此可见，元规则META001保证了先执行那些涉及到“create”的规则，然后再执行涉及到“delete”规则。即，安排了对规则例示的执行顺序。而META003排除了涉及到70的例示。通过元规则，我们可以用为-1的实用性（UTILITY）来删除与所要解决的问题无关的规则；或用实用性1把推理转移到某些特定的目标而不需改变目标规则在规则集里的顺序或事实在知识库中的顺序。

元规则控制模式的优缺点：

·本方法与前面所讨论的策略的不同之处是：在匹配阶段，其LHS被激发的规则的RHS先不被执行，它们的例示全被收集起来，直到在选择阶段被选上时才被执行。

·把控制性的信息清楚地从规则中分离出来，这极大地增加了知识库系统的适应性。此外，可以用元规则来控制、改变某些决策系统的推理策略，比如在ops5中，它的控制策略是从冲突集中选择时间标签最大的那条例示来执行，如果这种方法不适用于某些应用问题，本推理器只要简单地补充定义、描述其他一些元规则即可，不必改变原有的目标规则。

·在此方法中，检测冗余的规则是毫不重要的，因为每条规则只是对知识库中的事实进行一次性匹配，以后，只需随着知识库的变化而修改相应的部分。网算法提高了更改例示的效率。

·但是，元规则控制的推理过程产生了额外开销，因为为了选择实用性最好的例示而必须计算和维护一系列有关信息。这一点对于那些在实际应用中其规则的执行顺序并不重要，或其规则总是要对知识库进行比较大的修改的应用情况来说，更是不理想的。

6. PICASSO对不确定性（uncertainty）知识的处理

决策支持系统通常要在不确定的和不完全的知识（来自不同的知识源）的基础上做出决策。另外，用于进行推理的不同规则也可能带来不同程度的确定性。比如，下面的例子牵涉到两条规则R1和R2：

（R1）：推出决策D正确的概率是0.8——实际测试证明R1是非常确定的规则。

（R2）：推出决策D正确的概率是 0.2——R2是一个单凭经验的比较粗糙含糊的规则，当没有很多知识可用时，可使用它。

如果规则R1和R2两者都可用，我们说D在这种情况下具有大约75%的确定性，因为规则R1比R2可靠得多：即，由R1提供的证据应该比由R2接收的证据得分多（可靠）。此外，经常有这种情况：使用某一条规则所要做的实际观察是不可行的，或要花费昂贵的代价才能得到证据（比如病人要为某种医疗测试付出昂贵的费用），此时我们只好被迫采用可靠性不那么高但费用较少的决策，比如R2，当然，这仍然比丢骰子凭运气好得多。

大多数现有的专家系统的外壳（看〔WAT 85〕所做的调查）使用所谓单值的方法来做不确定性的推理；即，他们给每一个断言(事实)分配一个概率值。但是，它不可能表达规则和断言的可靠程度，所以单值法不适于决策支持系统。比如上面的例子分配给规则R1和R2的概率分别是0.8和0.2，这是单值的，它们不能表示可靠性的不同程度。此外，单值法对表示以下事实也有问题：我们对断言P一无所知；在大多数情况下未加定义地认为“一无所知”就是概率为0.5，这就造成了知识表达的严重问题。

为此，我们选择了二值法来表示模糊断言，称之为区间法，用以做为基本的模式。由于若干研究机构(〔APB 85〕，〔GIN 84〕和〔WIE 85〕)对二值法的研究结果，克服了单值法的缺点，这种模式已被推广。这种区间法和Dempster Shafer的证明理论是一致的；就是说，Dempster Shafer的理论可做为这种方法的理论基础。我们的区间法具有以下特征：

在这个方法中，相信某个命题P为真（true），主要通过分配一个区间值〔a b〕来对P进行估计，其语义如下：

i）P为真的概率至少是a；P的肯定度（confirmation）是a：

$$conf(\,[a\ \ b]\,) = a$$

ii）P为假的概率至少是（1-b）；P的不肯定度（disconfir mation)是（1-b）：

$$disconf(\,[a\ \ b]\,) = 1-b$$

iii）我们相信命题P的不确定（uncertainty）用（b－a）来计算：

unc（〔a b〕）＝b－a

iv）我们相信命题P的确定度（certainty 用1－（b－a）来计算：

cert（〔a b〕）＝1－unc（〔a b〕）

v）我们相信命题P的平均值（mean value）用 $\frac{a+b}{2}$ 来计算：

$$mv(〔a\ b〕) = \frac{a+b}{2}$$

比如，我们分配一个区间概率〔40 99〕给命题P，其意义如下：P的肯定度是40%，不肯定度是1%。即，40%的概率分配给了P，1%的概率分配给了（not P）；剩下的51%的概率还不知道如何分布：我们不知道这些概率中还有多少分配给P，有多少分配给（not P）；不确定度是59%。我们可用区间概率〔0 1〕表示对命题P一无所知（unknown）的情况：它的肯定度和不肯定度都是0；它的确定度是100%。

然而，区间概率不仅可用来描述模糊断言，也可用来描述模糊规则，请看以下的句法格式：

（if 〈条件集〉 （then（infer 〈断言〉〔with〈区间概率〉〕））

例9：（if（tall ?x）（then（infer（strong ?x）with（0.5 0.9））））

这条PICASSO规则表示：如果“某人是高的（tall）”这个条件为真，那么“他是强壮的（strong）”这个结论成立的概率至少为0.5，至多为0.9。

通常，在规则的LHS要对一个断言的确定因素（certainty factor）进行取反（否定），这可用命令negcert来实现：

（negcert〈断言〉）

另外，在知识库中，分配给某个断言的确定因素可用命令setcert来改变；可用命令getcert来读取：

（getcert〈断言〉）

（setcert〈断言〉）

为了在PICASSO中进行模糊推理，必须解决以下问题：

a）规则的LHS常常由一系列的条件组成，它们之间用and（与）、or（或）和not（非）运算符联结。我们需要一个算法来计算LHS的整个确定因素。

b）要提供一个决策标准，它决定什么时候（条件满足到什么程度）才对规则的RHS进行计算；也要提供一个函数，它计算已给出的、被推理的断言的确定因素。

c）给出一个组合算法，它把由不同的规则为同一个断言所提供的推理证据组合在一起。

6.1 计算整个条件部分（LHS）的确定因素

为了计算一个规则的LHS的不确定区间概率I，我们必须用逻辑运算符and、or或not对LHS的每个条件的不确定区间概率I1，……In进行运算。

如果我们必须计算A∧B的确定因素，我们必须知道A和B的关系：是否A和B是正相关，不相关或负相关的，要不然，计算结果会导致明显错误或不可靠的结果。令人吃惊的是，大多数专家系统用模糊逻辑（请看〔KAN 86〕和〔ZAD 85〕）的and/or运算符来

求解LHS，它们假设A和B为最大正相关（相关因子=1）。虽然这种假设能得到好的理论特征（比如，模糊（fuzzy）运算的运算符and/or满足于分配率）。但是，如果LHS的条件是不相关的或是负相关的，它可能导致大的计算错误。

有的研究课题（请看〔APB 85〕）使用区间法，但并不对A和B的关系进行任何假设；即，分别计算最低和最高的结果区间概率边界，然后选择相关因子使得最低边界成为最小值，而最高边界成为最大值。由于这种方法不使用任何涉及A和B关系的知识，会得到某些非常不可靠的区间概率。此外，这种手段经常计算一个区间概率的最低和最高边界，例如在这两个计算中有不同的相关，这将令人怀疑这种方法的总体一致性。

往往，专门领域的专家对所用的规则中的断言的相关性心中有数。因此，我们选择一种方法，允许PICASSO的程序设计员为规则分配一个相关因子，如果他有这方面的知识的话。如果他没有这方面的知识，系统将用省缺的相关值（相关值=0）来对LHS进行计算。

为了计算P（A1∧A2…∧An），我们把以下5种情况区分开来：

情形1：相关系数=1

Ai（i=1，n）为最大重叠（最好情形）——Ai之间有高度的正相关。

P（A1…∧…∧ An）=min {P（A1），…，P（An）}

情形2：相关系数=0

Ai为统计不相关（≈平均情况）——Ai为不相关。

P（A1…∧…∧An）=P（A1）×…×P（An）

情形3：相关系数=－1

Ai为最大不相交（最坏情形）——Ai之间为最大负相关。

P（Ai…∧…An）

=max {0，1－（1－P（A1）－…－（1－P（An））}

=max {0，1－P（A1）－…－P（An）}

注释：and（与）运算的最好情形是or（或）运算的最坏情形。

情形4：0＜相关系数＜1

用情形1和情形2得到的结果来进行插值运算。

情形5：－1＜相关系数＜0

用情形2和情形3所得到的结果来进行插值运算。

分别运用上面的公式对最低和最高边界进行运算，就可以容易地把以上的情况扩充到区间法；比如，相关系数=0，我们将得到以下结果：

（and（0.4　0.9）（0.8　0.9））=（0.32　0.81）

假设相关系数=1，我们会得到以下结果：

（and（0.4　0.9）（0.8　0.9））=（0.4　0.9）

区间概率的取反也可用以下公式计算：

（not（I））=not（（a　b））=（1－b　1－a）

我们可以很容易地用and（与）运算的公式来表达or（或）运算：

$$P(A \vee B) = P(A) + P(B) - P(A \wedge B)$$

当规则的LHS的条件所牵涉到的事实（断言）并未存在于识知库时，就提供了一个特殊

的问题。对这种情况有两种解释：

i）封闭性假设（closed world assumption）：

假设P为假：（〔0 0〕）。

ii）开放性假设（open world assumption）：

假设P为真：（〔0 1〕）。

大多数基于规则的程序设计语言采用第一种解释，而PICASSO采用第二种解释。

6.2 PICASSO的假言推论（modus ponens）

假设有下面这样一条规则：

（R1（if E）（then（infer H with〔c d〕））

如果规则R1的LHS诸条件的综合确定因素已算出(用6.1所描述的方法)为〔a b〕，现在提出的问题是，怎样利用规则提供的正／负证据进行断言H的计算？我们将规则R1简化为：

if E then H 〔c d〕

E〔a b〕

关于H的区间概率〔?1 ?2〕的概率分布是什么？

计算区间概率〔?1 ?2〕的函数称为假言推论生成函数（modus ponens generating functions），本节将讨论不同的假言推论函数，它们与区间概率法是兼容的。比如，本节开始所提到的那条PICASSO规则表达了以下含义："如果已知E为真，那么H的概率至少为c，至多为d"（然而，对于E为假或仅仅是部分为真的情况，就不能做这样一种命题式的解释了。）这种解释有两种不同的扩充情况，有可能用它们来处理PICASSO规则LHS的模糊断言：

i）P（H | E）的值至少为c，至多为d。

ii）P（H→E）的值至少为c，至多为d。

选择不同的解释法可推导出不同的假言推论生成函数，下面分别讨论它们。

6·2·1 条件概率的解释

一般假设：if E then H〔c d〕表达以下含义：P（H／E）的概率至少为c，至多为d。运用这种解释法，我们可推导出以下的假言推论规则：

用：if E then H〔c d〕

E〔a d〕

$$\frac{a+b}{2}>\theta \wedge (b-a)<\psi \qquad *5$$

*5：使用这种逻辑表达式是因为我们只知道E和H的关系，但并不知道not（E和H的）关系。其中θ和ψ是实际推导出来的常数，比如θ = 0.55，ψ = 0.85。

推出：H〔?1 ?2〕

其中，?1 = min（c*a+（1-d）*（1-a），

c*b+（1-d）*（1-b））

?2 = min（1，max（d*a+（1-c）*1-a），

d*b+（1-c）*（1-b））

推理：设E′表示怀疑E为真的观察资料，而$\bar{E}$表示E的后验概率，它是从E′推出来的，那么可推出以下公式：

$$P(H/E') = P(H \wedge E \mid E') + P(H \wedge not(E) \mid E') \qquad 式(1)$$

假如我们作以下的假设："如果我们知道E存在（或不存在），那么，与E有关的观察资料E′不再提供关于H的进一步的信息"——即，

$$P(H \mid E \wedge E') = P(H \mid E)$$

$$P(H \mid E \wedge not(E)) = P(H \mid (not(E))$$

于是从式（1）我们可推出：

$$P(E \mid E') * P(H \mid E' \wedge E) + P(not(E) \mid E')$$

$$* P(H \mid not(E) \wedge E') = P(H \mid E) * P((\bar{E})$$

$$+ P(H \mid not(E)) * P(not(\bar{E}))$$

$\bar{E}$表示E的后验概率——它是通过考虑另外的观察资料E′而被推出来的。但是，P（H | not（E））是未知的，我们只能说：如果E和H是正相关的，则P（H | E）>P（H | not（E））；如果E和H为负相关，则P（H | E）<P（H | not（E））。此外没有其他线索与P（H | not（E））有关，我们凭实际经验置P（H | not（E））为P（H | E）。这样做得到的误差的数量级取决于（1－P（$\bar{E}$））和 | correlation（E，H）|（E和H的相关系数的绝对值）。为了保持较小的误差，如果存在使E为真的正证据（比如，我们置θ＝0.55，ψ＝0.85）时，才能使用那些规则。

另外，考虑给予P（$\bar{E}$）一个区间概率〔a b〕，我们用$I_{\bar{E}}$来表示它。可通过计算最低边界（最小值）和最高边界（最大值）来求出P（H）的区间概率$I_{\bar{H}}$（其区间概率值随X的变化而变化，$I_{\bar{E}}$的a≤X≤$I_{\bar{E}}$的b）。或者我们可以凭实际经验置$P(\bar{E}) = \frac{a+b}{2}$。如果我们用第一种手法，另外还考虑不等式C≤P(H／E)≤d成立，于是我们可推出前面所描述的关于H的区间概率（后验区间概率〔?1 ?2〕）。

另一种由PROSPECTOR专家系统（〔DUD 79〕）选择的方法是：要求专家也提供与not（E）及H有关的知识P（H | not（E））。显然，在这种情况下，使用前面介绍的公式不会有误差。由于这种方法假定两种关系都会给出，所以无论P（E）的概率是什么，PROSPECTOR的规则都会被激发、执行。然而，PICASSO的规则只有在P（E）至少为0.5时才会被激发、执行。可是，PROSPECTOR的方法有严重缺陷：虽然专家能提供确切的有关P（H | E）知识，却未必能提供P（HInot（E））。

让我们给出一个常识性的例子来说明这个要点：

显然，P（街道湿 | 下雨）＝1，因为下雨使街道湿了。但你要想估计

P（街道湿 | not（下雨））

即，P（街道湿 | 不下雨），那就会有很大问题。使用PICASSO时，如果与not（E）和H有关的知识是可知的，你应使用这种规则：if（not E）then H。

6·2·2 或概率（Probability of a disjunction）的解释

一般思想：这种方法对if E then H〔c d〕做如下解释：P（E⟶H）的概率至少是c，至多为d，即，这种方法赋予的不确定（uncertainty）区间概率值为：

$$P(E\longrightarrow H)=P(\neg E\vee H)=1-P(A)+P(B)-P(\neg A\wedge B)$$

一般说来，使用假言推论可能会导致不一致性——特别是如果 mv（E）接近于0的时候。总之，为了防止得到不一致的结果，不同的假言推论函数的左边都包含了检查不一致性的断言；即，在整个推理过程中，我们排除掉那些会把应用引到不一致的、由规则提供的断言。

我们根据LHS和RHS之间的相关系数把问题区分为三种情形：

情形1：相关系数=1时，用模糊逻辑（fuzzy logic）or 函数

用：if E then H〔c d〕

E〔a b〕

$(1-a<c)\wedge(1-b\leqslant d)$

推出：H〔c d〕

情形2：相关系数=0时，用传统的概率理论

用：if E then H〔c d〕

E〔a d〕

$(a\neq 0\wedge(c+a\geq 1)\wedge(b+d\geq 1))$

推出：$H\left[\frac{c+a-1}{a}\quad\frac{b+d-1}{b}\right]$

情形3：相关系数=－1，即最大不相关

用：if E then H〔c d〕

E〔a b〕

$(a+c)\geq 1$

推出：H〔c+a－1 b+d－1〕

以上三种假言推论生成函数的证明：

情形1：

从max（1－P（E），P（H））≥C

可推出：if(1－a<c) then P（H）≥c

要不然，“推导不出任何关于P（H）的东西”。

对于max（1－P（E），P（H））≤d，我们可推出以下两种情况：

if（1－b≤d）then P（H）≤d

else “我们会陷入不一致”。

情形2：

$$P(H)\geqslant\frac{C+P(E)-1}{P(H)}=1-\frac{1-c}{P(E)}$$

$$P(H) \leqslant \frac{d+P(E)-1}{P(E)} = 1-\frac{1-d}{P(E)}$$

右边的两个表达式依赖于P(E)而单调地增加。因此，我们令P(E)=a，得到最低边界，而令P(E)=b，得到最高边界。在综合考虑了一致性之后，我们得到前面情形2的推理条件和结果。

情形3：

$$1-P(E)+P(H) \geqslant C$$

$$P(H) \geqslant C+P(E)-1$$

此外，

$$P(H) \leqslant d+P(E)-1$$

取最坏的情形(P(E)=a)做为最低边界，取最好的情形(P(E)=b)做为最高边界，排除不一致性问题，我们得到〔a+c-1　b+d-1〕。

PICASSO运用在6·2·1讨论的假言推论函数做为其省缺策略；但是，用户可以通过在调用推理器时设置一定的开关来选择其他的三种组合。

6·3 证据的组合

在实际应用中，若干条规则可能为同一个断言P提供有关的正、反证据。假定有n条规则都与断言P有关，它们的区间概率I1，···In已经分别用在6·2讨论的方法推算出来，那么P的总确定因素必须通过把由区间概率I1，···In所表达的证据组合起来进行计算。简言之：

R1为断言P提供区间概率I1=〔a1 b1〕的证据

· · · · · ·

Rn为断言P提供区间概率In=〔an bn〕的证据

支持P的那些证据的总区间概率I=〔?1 ?2〕

PICASSO使用了两种不同的组合函数：

·函数dscomb做以下假设：来自不同知识源的证据是互不相关的。在互不相关的假设前提下，如果已找到较为确切的证据，则函数dscomb会提高P的总概率。

·函数sscomb做以下假设：证据是通过解释同一个知识源而得到的；即，不同的区间概率代表了对同一个问题的不同“选择”。因此，sscomb函数使用了带权的投票法，它用不确定性(uncertainty)来权衡要被组合的每一个区间概率：确定性较高的区间概率得到较大的权，不确定的区间概率得到较小的权，因此，较为确切的证据将不会引起总概率的提高。

比如有下面的两个式子，请看它们的差别：

dscomb(〔0·9　1〕〔0·9　1〕)=〔0·99　1〕

sscomb(〔0·9　1〕〔0·9　1〕)=〔0·925　0·975〕

(sscomb是通过平均值mv和不确定值unc来计算的)。

前者的两个证据的区间概率均为〔0·9　1〕，是相当确切的证据，它们来自不同的知识源，且互不相关经，使用函数dscomb组合之后的总区间概率为〔0·99　1〕，总概率提高了许

多，说明总证据更确切了。后者的两个证据的区间概率也都为(0.9　1)是相当确切的证据，但这些证据是通过对同一个知识源的不同解释而得到的，使用函数sscomb组合后的总概率为〔0·925 0·975〕，提高不大。

函数dscomb的计算公式如下：

$$\text{dscomb}([l_1\ u_1],[l_2\ u_2]) = \left[\frac{l1\times u2+l2\times u1-l1\times l2}{1-l1\times(1-u2)-l2\times(1-u1)}\quad\frac{u1\times u2}{1-l1\times(1-u2)-l2\times(1-u1)}\right]$$

我们用以下的方程式来定义函数sscomb：

$$令 t=\text{unc}(I1)+\text{unc}(I2)$$

$$\text{mv}(\text{sscomb}(I1,\ I2)) = \begin{cases} 0.5 & t=2 \\ \dfrac{\text{cert}(I1)}{\text{cert}(I1)+\text{cert}(I2)}*\text{mv}(I1)+\dfrac{\text{cert}(I2)}{\text{cert}(I1)+\text{cert}(I2)}*\text{mv}(I2) & 1<t<2 \\ \dfrac{\text{unc}(I2)}{\text{unc}(I1)+\text{unc}(I2)}*\text{mv}(I1)+\dfrac{\text{unc}(I1)}{\text{unc}(I1)+\text{unc}(I2)}*\text{mv}(I2) & 0<t<=1 \\ \dfrac{\text{mv}(I1)+\text{mv}(I2)}{2} & t=0 \end{cases}$$

$$\text{unc}(\text{sscomb}(I1,\ I2)) = \begin{cases} 0 & t=0 \\ \dfrac{\text{unc}(I1)*\text{unc}(I2)}{\text{unc}(I1)+\text{uuc}(I2)} & 0<t<=1 \\ \text{unc}(I1)*\text{unc}(I2) & 1<t<2 \\ 1 & t=2 \end{cases}$$

假设其组合概率为〔?1　?2〕

通过平均值mv和不确定值unc就可以计算出〔?1　?2〕：

$$\begin{cases} \text{mv}=\dfrac{?1+?2}{2} \\ \text{unc}=?2-?1 \end{cases}$$

这两个算法有以下特点（详细的讨论请看〔Eick 87a〕）：

·在这两个函数中，所得到的组合证据的不确定性比被组合的各个证据的区间概率的不确定性要小；因为证据的线索多了，便增加了其可靠性。

·dscomb(〔0 1〕〔a b〕) = sscomb(〔0 1〕〔a b〕) = 〔a b〕；

完全不确定（如区间概率为〔0 1〕）的知识不会改变总体的证据。

·dscomb满足可结合率；对 sscomb 来说，当不确定性小于等于 0.5时的组合区间概率是满足结合率的。

·在对区间概率进行组合时，sscomb分配较高的权给较为确定的区间概率；即，它把

优先权赋给这样的证据——这些证据来自于有较高确定性的规则。而dscomb并不完全具有这些特点。

• dscomb和sscomb满足交换率。

PICASSO用dscomb做为省缺组合函数；然而，对于某些特定的断言，可指示推理器使用函数sscomb。

6·4 一个小例子

下面我们给出一个小例子，说明如何使用前面讨论的函数。考虑有下面的 PICASSO 规则集：

```
( ( r1 ( if ( and ( cloudy-sky ) ( humid ) ) )
       ( then ( infer ( rain ) with ( 0.4 0.9 ) ) )
       ( corr 0.5 ) )
  ( r2 ( if ( high-pressure ) )
       ( then ( infer ( rain ) with ( 0.0 0.7 ) ) )
  ( r3 ( if ( and ( hot ) ( humid ) )
       ( then ( infer ( rain ) with ( 0.6 1.0 ) )
       ( corr 1.0 ) )
  ( r4 ( if ( not ( high-pressure ) )
       ( then ( infer ( rain ) with ( 0.3 1.0 ) ) ) )
```

其中，corr是correlation的缩写，表示相关系数；若未给出corr，隐含corr = 0。

我们给出以下三组测试数据，然后调用PICASSO 推理器来进行计算，在计算过程中使用了下面的函数：

• 用§6.1介绍的方法分别计算各条规则LHS的总区间概率。

• 用§6·2·1介绍的假言推理函数（其中θ = 0·55，ψ = 0·85）分别计算出各条规则RHS关于rain（下雨）的概率。

• 由于r1、r2、r3和r4的结论都是关于 rain 的，且未加特别说明，隐含着来自不同知识源，故用dscomd函数计算关于rian的组合概率。

下面是给出的三组数据和推导的结论：

推导下雨的前提	数据1（概率）	数据2（概率）	数据3（概率）
cloudy—sky	(0.88 0.90)	(0.60 0.62)	(0.90 0.92)
humid	(0.88 0.90)	(0.58 0.60)	(0.62 0.64)
hot	(0.80 0.82)	(0.90 0.94)	(0.65 0.67)
high—pressure	(0.32 0.34)	(0.80 0.82)	(0.90 0.92)
结论：r1（rain）	(0.35 0.86)	not fired	(0.28 0.78)
r2（rain）	not fired	(0.05 0.76)	(0.02 0.73)
r3（rain）	(0.48 0.89)	(0.35 0.76)	(0.37 0.78)
r4（rain）	(0.20 0.90)	not fired	not fired
总结论：（rain）	(0.67 0.84)	(0.32 0.64)	(0.40 0.60)

在推理结束之后，对那三组数据的运行结果我们可以做以下解释：

·从第一组数据推出的rain（下雨）的总概率为（0·67 0·84），这种可能性是较大的，其平均概率达到75%左右；后两组数据关于rain的总概率分别为（0·32 0·64）和（0·40 0·60），平均概率达50%左右。但是，第二组数据推出的总概率是相当不可靠的，因为不确定性达到0·32。

·虽然我们从第一组数据中推出的r1、r3和r4关于rain的区间概率都相当不确定（分别为（0·35 0·86）、（0·48 0·89）和（0·20 0·90）），但它们是支持rain这个结论的三个独立存在的事实，故导致了相当可靠（75%的平均可能性）的关于rain的结果概率。

7. PICASSO的用户接口

PICASSO推理器被启动之后，它会在知识库的基础上，对活动规则集进行推理，然后回送所有的输出变量做为计算结果。其输出结果用一个相关表来表示。可以把活动的PICASSO规则集看做为函数，此函数的输出变量结果取决于输入变量的值。

大多数推理过程按以下三步来进行：

i）初始阶段，此时，要求通过人机对话方式提供某些必需的信息，而且把输入的信息转换成更适合于推理过程的格式。

ii）主推理过程

iii）终止推理过程，其间，把推理结果转换为适合于程序调用者的数据结构。

把这三个阶段分离开来：把主推理过程与带有应用程序的推理器的通讯分离开来，这就改善了知识库系统的适应性。

PICASSO推理器通过三个规则集来支持上述三个阶段：初始规则集，主（活动）规则集和终止规则集。一旦启动了推理器，它就依照上面所说的次序来分别计算这三个规则集。

PICASSO提供两种版本。

i）PICASSO的解释版本。

ii）PICASSO的编译版本：用一个编译器把给定的PICASSO规则集转换成普通的LISP函数，这种方法有效地对给定的规则集进行推理（其结果取决于输入变量的值）。

7·1 解释版本

PICASSO的解释版本通过调用名为PICASSO的LISP函数来启动，它带有以下参数：

·一个主（活动）规则集。

·一个知识库。

·一个初始规则集，为任选项。

·一个终止规则集，为任选项。

·一个被动（储备）规则集：这里面的规则在过程的开始是不活动的，但在推理期间是可活动。

·一组输入变量，任选。

·一组输出变量，任选；如不提供这些变量，所得结果存入系统变量。

注意：

·为了防止全局变量带来的某些付作用，凡由PICASSO系统使用的全局变量以

&PICASSO—或$PICASSO—牵头；由PICASSO用户使用的全局变量以&或$牵头，希望与此无关的其他调用程序不要使用有这些特征的全局变量。

• PICASSO的全局变量可带单值或多值，其值可被PICASSO中的任何规则及其他的推理器直接使用。

7·2 编译版本

在编译版本中，一个给定的PICASSO规则集，或规则集的集合可被转化为LISP函数，它输入变量的值，在推理过程终止后，它把输出变量值做为计算结果。这个函数可作为普通的LISP函数；即，它可以象每个其他的LISP函数那样被编译和使用。

我们用以下的句法格式来定义编译版本的用户接口：

```
(ifengine
  (ie-name 〈推理器的名字〉)
  (act-rule 〈推理器的活动规则集〉)
  (init-rule [〈初始规则集〉])
  (ter—rule [〈终止规则集〉])
  (Pass-ruce [〈被动(储备)规则集〉])
  (schema 〈知识库的结构方案〉)
  (knowledge—base 〈知识库的名字〉)
  (input—v [〈输入的全局变量表〉])
  (output—v 〈输出的全局变量表〉)
  (local—v [〈局部变量及初始化〉])
    • • • • • • •)
```

注意：其中，函数ifengrne是PICASSO编译版本的名字。ie—name指定模拟推理器的LISP函数的名字。schema包含了知识库的结构模式，推理器将在其上面进行推理工作；schema用于检测被编译的规则集里的语义和句法错误。编译版本比不上解释版本灵活。但是，使用编译版本会有以下优点，这使它具有吸引力：

• 编译将在编译时而不是在运行时检查规则的句法正确性。

• 在编译时可进行大范围的错误检测：比如检查变量的正确使用，检查规则中用到的数据模式以保证所使用的断言的一致性等等，在编译时还可选择其他若干任选项。所有这些工作若要在运行时才进行将会导致很多错误。

• 易于并行地使用函数和基于规则的程序设计语言。

• 可以担保激发、执行规则不会在调用推理器的函数中引起付作用。

• 所产生的函数可被编译，速度会更快。

• 在产生编译代码时，可运用优化技术，特别是在模式匹配运算方面。

• 推理器可递归地调用它自己，它支持层次式的程序设计和基于规则的自上而下的方法，并简化基于规则的程序设计系统的模块。

8. 关于PICASSO工作步骤的简例

现在让我们通过一个例子，说明整个工作步骤。

例10：假设有一个大的温室（greenhouse）分成两部分（section）。每一个部分都有它自己的窗户（window）、加热器（heater）和喷水器（mister）。你的任务是写一个PICASSO程序，建议管理员什么时候开（open）、关（close）窗户，开（turu on）、关（turn off）加热器、喷水器或打电话给某人来修理坏了的设备。

我们采用以下几个步骤来解决这个问题。

a）用符号图形的数据模式来描述数据的类别、属性等结构方案。假如schema（方案）名为greenhouse。我们打算把问题规纳为两个类别：section（部分）和 environment（环境）。其中类别section包含以下属性：sect-number（部分号）、temperature-inside（内部温度）、window（窗户）、heater（加热器）、mister（喷水器）、temperature-desired（理想温度）、humidity-desired（理想湿度）和 accept-windspeed（可接受的风速）等。而类别environment有以下属性：

temperature-outside（外界温度）、humidity-outside（外界湿度）和windspeed（风速）等。

它们的特性和约束条件如下所述（注意：温度为华氏）：

schema	greenhouse
class	section
simple attribute	sect-number
property	unigue, onto
constrains	(between 1 and 2)
type	INTEGER
simple-attribute	temperature -inside
property	unique, onto
constrains	(between -100.0 to 100.0)
type	REAL
simple-attribute	humidity-inside
property	unique, onto
constrains	) 0.0 and (100.0
type	REAL
simple-attribute	window
property	unique, onto
constrains	open, close or broken
type	SEXPR
simple-attribute	mister
property	unique, onto
constrains	on, off or broken

type	SEXPR
simple-attribute	temperature-desired
property	unique, onto
constrains	> 75.(and <85.0
type	REAL
simple-attribute	humidity-disired
property	unique, onto
constrains	> 0.0
type	REAL
simple-attribute	accept-windspeed
preperty	unique, onto
constrains	> 0.0
type	REAL
class	environment
simple-attribute	temperature-outside
property	unique, onto
constrains	(between -130.0 to 10.0)
type	REAL
simple-attribute	humidity-outside
property	unique, onto
constrains	> 0.0 and <100.0
type	REAL
simple-attribute	windspeed
property	unique, onto
constrains	> 0.0
type	REAL

在编译时，PICASSO系统要用schema中的数据结构来对规则集里的规则以及知识库中的事实（断言）进行一致性检查。

b）定义初始规则集

一般，初始规则集的内容主要是对知识库初始化和置初值，做为后面进行推理的起码事实依据。假如用户现在手头有以下一组数据：

class section;

sect-number	1
temperature-inside	74.00
humidity-inside	0.4
wimdow	close
heater	on
mister	broken

```
      temperature-desired                70.0
      humidity-desired                   0.7
      accept-windspeed                   20
class section;
      sect-numben                        2
      temperature-inside                 72.0
      humidity-inside                    0.5
      window                             open
      heater                             broken
      temperature-desired                71.0
      humidity-desired                   0.6
      accept-windspeed                   15
class environment;
      temperature-outside                71.0
      humidity-outside                   0.65
      windspeed                          10
```

根据这些数据，对知识库进行初始化，即用create命令在知识库中生成带有上述数据的实体（遵循schema给出的数据结构）：

```
(setq &initial-rules
      '((rule)(perform
                (create section((sect-number  1)
                (temperature-inside          74.0)
                (humidity-inside              0.4)
                (window                     close)
                (heater                        on)
                (mister                    broken)
                (temperature-desired         70.0)
                (humidity-desired             0.7)
                (accept-windspeed              20)))
       (create  section((sect-number                    2)
                        (temperature-inside            72)
                        (humidity-inside              0.5)
                        (window                      open)
                        (heater                    broken)
                        (temperature-desired         71.5)
                        (humidity-desired             0.6)
                        (accept-windspeed              15)))
       (create environment
```

```
( ( temperature-outside        71.0 )
  ( humidity-outside           0.65 )
  ( windspeed                   10 ) ) ) ) ) ) )
```

其中，setq是LISP内部函数，表示后面跟着’号的所有表内容都赋给全面变量 &initial-rules。

c）定义活动规则集

根据例10要解决的问题，我们把解决这些问题的方法用规则的形式表示出来：

```
(setq &rules
    '( ( rule1 ( if ( section ? entity-num ? sec )
                    ( window ? entity-num open )
                    ( windspeed ? ent2 ? wind-out )
                    ( accept-windspeed ?entity-num ?windspeed-acceptable)
                    ( %greaterp ? wind-out ? windspeed-acceptable ) )
               ( then ( conclude close-window-in ? sec ) ) )
       rule2 ( if ( section ? entity-num ? sec )
                  ( heater ? entity-num ? heat )
                  ( temperature-inside ? entity-num ? temp-in )
                  ( temperature-desired ? entity-num ? desired-temp )
                  ( %lessp ? temp-in ( difference ? desired-temp 20 ) )
                  ( %member ? heat ( heater broken ) ) )
               ( then ( conclue Notify Bill smith at phone No. 666-8211 to
                        repair the broken heater ) ) )
                ......
       )
```

用LISP函数setq把’号后面的表内容全赋给了全局变量 &rules。

d）调用用户接口

假如要使用用户控制模式(user-model)的推理策略，并设知识库（knowledge-base）的名为main，根据用户接口的要求定义上面的相关表（用户接口）：

```
(setq ass' ( ( ie-name user-model )
             ( knowledge-base main )
             ( schema greenhouse )
             ( init-rule &initial-rules )
             ( act-rule &rules ) ) )
```

然后调用PICASSO推理器infengine：

```
             ( infengine &ass )
```

9. 参考资料：

ABR 74

Abrial,J.R.: *Data Semantics*;*Proc.IFIP TC2 Conference, North Holland*;*Cargese*(1974).

AKG 85

Avni,E.;*Kandel,A.*;*Gutpa*:*Soft relational data base in CAD/CAE and expert systems*;*pp.*573-591;*in*〔*GUP* 85〕.

APB 85

Appelbaum,L.;*Ruspani,E.*:*ARIES*:*An Approximate Reasoning Inference Engine*; *in* 〔*GUP* 85〕;*pp.*745-765.

BFKM 85

Browston,L.;*Farell,R.*;*Kant,E.*;*Martin, N.*:*Programming Expert Systems in OPS5*; *Addison Wesley*(1985).

BOR 85

Bordiga,Alexander:*Language Features for flexible handling of exceptions in information systems*;*ACM Transactions of Data Basc Systems*(*Dec.* 1985).

BUC 84

Buchanan,B.;*Shortliffe,E.*: *Rule Based Expert Systems*;*Addison Wesley*(1984).

COH 85

Cohen,Paul:*Heuristic Reasoning about Uncertainty An Artificial Intelligence Approach*;*Pitman*(1985).

DUB 85

Dubois,D.;*Prade,H.*:*Combination and Propagation of Uncertainty with Belief-functions- a Reexamination*; *Proc.IJCAI Conf*,*pp.*111-113,*Los Angeles*(1985).

DUD 79

Duda,R.;*Gaschnig,J.*;*Hart,P.*:*Model Design in the PBOSPECTOR Consultant System for Mineral Exploration*;*in* *"Expert Systems in the Micro Electronic Age" edited by D.Michie,Edinburgh Press*(1979).

EICK 84

Eick;*Christoph F.*:*From Natural Language Information Requirements to Good Data Definitions—a Database Design Methodology*;*Proc.First Data Engineering Conference*(*Los Angele*:1984).

EICK 85

Eick,C.F.;*Lockemann,P.*:*Acquisition of Terminological Knowledge using Database Design Techniques*;*Proc.Int.Conf.on Management of Data*(*Austin* 1985).

EICK 87*a*

Eick, Christoph F.: *Data Management for LISP Knowledge Bases*;*submitted for publication*(*July* 1987).

EICK 87*b*

Eick,Christoph F.: *Combining Evidence for Computerized Guessing*;*internal paper*

(*January* 1987).

FOR 82

*Forgy,L.;Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem;Artificial Intelligencc,pp.*17-87(*Sept.*1982).

GIN 84

*Ginsberg, M.L.:Non-monotonic reasoning using Dempster's rule;Proc.AAAI-*84; *pp.*126-129,*Austin*(1984).

GUP 85

Gupta,M.;Kandel,Brandler,Kiszka: Approximate Reasoning in Expert Systems;North Holland(1985).

KAN 86

Kandel , Abr. : Fuzzy Mathematical Techniques with Applications , Addison Wesley (1986).

LOOPS 83

*Bobrow,D.;Stefic,M.: The LOOPS Manual;Xerox Cooperation(Dec.*1983).

MBW 80

Mylopoulos,J.;Bernstein;Wong, K.T.: A Language Facility Designing Interatcive Database Intensive Systems;ACM Transactions on Database Systems(June 1980).

McL 78

*McLeod,Dennis:A Semantic Data Base Model and its associated structured user interface;PhD-dissertation;MIT/LCS/TR*214(1978).

SHA 76

Shafer,G.: A Mathematical Theory of Evidence;Princeton University Press,Princeton(1976).

SHU 76

*Schubert,L.:Extending the expressive power of semantic networks;Artifical Intelligence,pp.*163-198(1976).

THO 85

*Thomson, Terence: Parallel formulation of evidental-reasoning theories ;Proc.IJCAI Conf.,pp.*321-327,*Los Angeles*(1985).

TRI 85

Trillas,E.;Valverde, L.: On Mode and Implication in Approximate Reasoning;in [*GUP* 85] ;*pp.*157-166.

WAT 85

Waterman: A Guide to Expert Systems;Addison Wesley(1985).

WIE 85

Wierzchon,S.T.: Mathematical Tools for Knowledge Representation;in[*GUP* 85]; *pp.*61-70.

YU 86

*Yu,Yan: Reasoning in Uncertainty in Rule Based Expert Systems;Master Thesis,University of Houston(Dec.*1986).

ZAD 85

Zadeh,L.A.: The Role of Fuzzy Logic in the Management of Uncertainty in Expert Systems;in [*GUP* 85];*pp.*3-31.

SEL 84

Sell, peter S.: Expert Systems—A Practical Introduction.

CHA 87

Chang, Cha Liang: Automatic Generation of Control Strategies For Rule Based Expert Systems,

Master Thesis, University of Houston (*Dec.* 1987).