第4卷 第5期
1993年10月

软 件 学 报
JOURNAL OF SOFTWARE

Vol. 4, No. 5
Oct. 1993

LogC 智能推理语言和环境

冯玉琳　　黄　涛　　王太权

（中国科学院软件研究所，北京 100080）　（中国科学技术大学，合肥 230026）

LogC: A LANGUAGE AND ENVIRONMENT FOR KNOWLEDGE INFERENCES

Feng Yulin and Huang Tao

(*Institute of Software, Academia Sinica, Beijing* 100080)

Wang Taiquan

(*University of Science and Technology of China, Hefei* 230026)

Abstract The LogC language and environment is designed to support knowledge inferences for intelligent problem solving. This paper describes the LogC motivatives and some of design features.

摘要 LogC 语言和环境面向知识推理，旨在为包括专家系统在内的一大类智能软件系统设计提供一般目的的集成化环境支持. 本文从知识表示、推理效率和环境结构化等三方面阐述 LogC 语言和环境的设计特点.

§0. 引　言

智能软件以知识处理为核心，具有许多不同于一般应用程序的特点. 以强推理能力为特征的智能推理语言和环境，自然也就应具有许多不同于一般程序设计语言和环境的特点. 知识表示、推理效率和软件工程界面是设计智能推理语言和环境时需重点考虑的三个要素.

面向规则的计算模式与人的认识模型有着很好的一致性. 在知识的表示和推理方面，面向规则的计算模式有着很大的优点. 但是，知识的推理过程常常伴随有一些特别的计算. 显然，对于这样的计算的处理，面向过程的计算模式与面向规则或者逻辑的计算模式相比较，当然要方便得多. LogC 使这两种计算模式密切耦合，给知识表示带来方便.

启发式推理是提高智能语言推理执行效率的核心所在. 基于消解的 PROLOG 推理机

本文 1991 年 3 月 22 日收到，1991 年 6 月 13 日定稿. 本研究课题由国家自然科学基金和国家高技术发展计划资助. 作者**冯玉琳**，研究员，现任中国科学院软件研究所副所长，主要研究领域为系统模型和语义理论，软件设计方法学和软件工程环境等. **黄涛**，现为在读博士生，主要研究领域为并发模型理论，软件工程环境，系统软件. **王太权**，助教，1989 年硕士毕业于中国科技大学，主要研究领域为软件工程环境，系统软件.

制是说明性推理的例子，其总体控制策略只能是遍历整个推导树，很难加入启发性信息. 与此不同，LogC 按过程性推理方式进行推理，视推导为一种问题求解的过程，控制策略由规则集中给出的一些元控制组合决定，用户可以通过定义启发性优先函数等机制使 LogC 推导力求沿着推导树的优化路径进行.

近十年来，随着 UNIX 的发展，基于 UNIX 和 C 的软件资源愈来愈丰富，世界上从事智能软件设计的专家们已充分认识到开发基于 UNIX 和 C 的 AI 程序设计环境的重要性. LogC 包含 C 作为基语言，与 C 全兼容，各种 C 应用程序和库程序都可原封不动地在 LogC 中使用. LogC 环境基于 UNIX 开发，可以最大限度地利用已有的基于 C 和 UNIX 的各种软件资源.

本文从知识表示、推理效率和环境结构化等三方面阐述 LogC 语言和环境的设计特点.

§ 1. LogC 语言特点

LogC 语言是一个多模式的面向知识处理的系统程序设计语言. 函数是程序结构的组成单位，表示作用于具体数据元的过程性知识. 与此同时，LogC 允许象定义函数一样去定义一个推导规则集，表示作用于一类数据元的推导. 面向函数的计算与面向规则的推导，两者有机结合，不需要任何繁冗的程序接口.

LogC 包含 C 语言作为基语言，按照 C 语言的语法定义表达式和函数.

LogC 规则集是进行有关知识推理的程序结构单位，是 LogC 程序设计的核心. LogC 规则集包含有三类知识：静态知识、动态知识和元控制，分述如下：

静态知识是不受程序执行影响的永久性知识，由一组 IF～THEN 型的推导规则描述. 规则的条件部分是一串条件元的合取，每个条件元均是 C 语言表达式. 规则的动作部分是一动作序列，每个动作是 C 语句，或者是用于数据空间更新的特殊操作语句.

动态知识是随程序的执行环境而变化的临时性知识，存放在数据空间中. LogC 推理的数据空间由类组成，类空间是相同类型的数据即类元的集合. 类元可以是初始化时已有的，也可以是在推理过程中生成的. 规则集定义中，与该规则集相关联的类空间由类元变量指明.

第三类知识是推理元控制信息，在规则集首部说明，用于控制规则集的推导过程. LogC 提供的多种元控制策略，可灵活地组合使用. 常用的一些元控制说明如下：

• 选择模式

选择模式由选择元控制[selection mode]指定，用来消除匹配冲突. LogC 提供三种选择模式：bydata 选取含最优先类元的规则类元对；bytime 选取含有最近引用类元的规则类元对；byrule 则按规则的自然顺序选取第一个匹配的规则和与之匹配的最优先类元对.

• 优先说明

优先说明决定类元的优先关系. LogC 提供两种优先关系元控制，即优先函数指定[priority type:function]和类优先次序指定[priority type1>type2]. 前者指定函数 function 为 type 类的优先函数，这里的 function 是用户定义的整型 C 函数；后者表示 type1 类优先于 type2 类.

• 回溯

LogC 规则集推导的回溯工作方式由回溯元控制[backtracking n]给定，这里 n 是回溯宽度，表示从每个数据空间状态出发最多选择 n 条不同路径进行推理. 当 n 省缺时为全回溯.

· 推理循环遏制

以数据空间的状态为结点，推理过程中所有可能达到的状态结点在逻辑上构成一棵推理树. 当推理树上相同结点重复出现时，推理可能陷入死循环. 在进行启发式推理时，数据空间的改变经常是可逆的，死循环出现的概率很大. 为了能有效地支持启发式推理，一个成功的推理机制应能尽量避免推理死循环. LogC 使用特征值计算法检查是否出现重复的数据空间状态或者类元，提供重复状态检测[statecheck type]和重复类元检测[classcheck type: level]两种推理元控制.

LogC 还提供有其他一些元控制，如预测推理、步进控制等，这里不再一一列举.

使用 LogC 元控制可以很方便地实现启发式推理. 比如，为指定类设计适当的 C 函数作为启发式优先函数，结合选择模式 bydata，可以实现最佳优先的搜索算法；如果在推理的每一步均清除当前所在类空间，即可实现局部择优算法；如果在优先函数中对推理路径长度加权计算，使类元优先数主要取决于该类元所在推理深度，则推理过程倾向于纵向搜索；反之，可使推理过程倾向于横向搜索.

§ 2. LogC 推理效率

LogC 推理是数据驱动的，是通过规则集作用于数据空间实现的. 其基本推理过程是周而复始的匹配—选择—动作的循环. 由匹配产生冲突集，根据选择策略从冲突集中选择一组规则类元对执行之.

在整个规则集推导过程中，匹配时间所占的份额最大，有时竟占整个规则集执行时间的 90%. 冲突集更新时，如果将规则集中的每一条规则与数据空间中的每一个类元都交叉进行匹配，显然，系统效率就显得非常低了. 为此，LogC 实现采用了改进的类 Rete 网匹配技术. 在动作执行过程中记录对数据空间的修改；在动作执行结束后，根据数据空间的修改记录更新冲突集. 这就避免了匹配过程中的大量冗余计算，提高了规则集的执行效率.

再考虑到 LogC 具有灵活的启发式推理控制机制，与其他基于规则的语言相比，LogC 在推理执行效率方面的优势是显而易见的，见表 1 和表 2.

表 1 是 LogC 和 PROLOG 求解九宫问题推理执行效率的比较. 随机取样 5 组不同数据，对解释执行效率比较，LOGCE 在 1 分钟内全部得到结果，而 C－PROLOG 只完成 1 项，其余计算时间均超过 5 分钟；对编译执行效率比较，LOGCC 对 5 组初始数据均能在 1 秒钟内计算完成，而 TURBO－PROLOG 在 5 分钟之内因时空限制仍只能计算出 1 项.

对这个例子使用 LogC 求解时，选择棋盘当前格局到终结格局各棋子位置的 Manhattan 距离和作为棋盘格局的启发式优先函数；但这一优先函数并不能保证对于所有的初始棋盘格局都能找到解，因此，在 LogC 推理中要加上回溯元控制；又由于棋子的移动是可逆的，于是又加上重复状态检测元控制，采用特征值计算法避免推理过程进入死循环. 这些元控制的组合作用可以确保从任意一个初始格局出发迅速找到一个可行解路径，而预测推理的使用则又会大大缩短推理步数. 显然，这一切都是 PROLOG 的推理机制所不能提供的，

即使勉强实现了，效率也是极低的.

表 1　求解九宫问题推理执行效率比较

初态	LOGCE 执行时间 (1/60 秒)	C−PROLOG 执行时间 (1/60 秒)	LOGCC 执行时间 (1/60 秒)	TURBO−PROLOG 执行时间 (1/60 秒)
8 7 6 5 4 3 2 1 0	659	×	12	×
1 3 5 2 4 6 7 8 0	617	×	12	×
4 1 3 6 0 5 8 2 7	1947	×	34	×
0 2 3 1 4 6 7 5 8	47	184	1	13
2 1 6 4 0 8 7 5 3	3056	×	55	×

表 2　启发式求解货郎担问题推理效率增益情况比较

初值	LogC					C	
	近优解	名次	误差率 (%)	LOGCE 时间 (1/60 秒)	LOGCC 时间 (1/60 秒)	最优解	C 时间 (1/60 秒)
7(city71)	2140.750	1/360	0	40	3	2140.7507	194
7(city72)	2825.291	2/360	8.6462	39	4	2600.4511	162
8(city81)	3013.830	15/2520	11.8018	67	6	2695.6901	1009
8(city82)	2855.022	3/2520	1.3622	67	6	2816.6556	935
8(city83)	2817.968	1/2520	0	64	6	2817.9683	756
10(city10)	26906.70	1/181440	0	161	15	26906.706	4314
20(city20)	4440.842	—	—	2412	263	—	> 45 分
30(city30)	5465.164	—	—	11424	1401	—	—

(注：以上表列时间均为在 PC386 上的实测时间)

表 2 从另一方面比较 LogC 的推理能力. 众所周知，求解货郎担问题的归约矩阵算法的时间复杂度是 $O(n^2 2^n)$，n 是城市数目. 当 $n>20$ 时，使用该算法计算的时间就非常可观了. 但由于 LogC 可充分利用启发式知识推理求解，例中的 LogC 算法选用最小支撑树长度作为启发性优先函数，可在不超过 $O(n^3)$ 的时间内求得一个近优解. n 值越大，LogC 使用启发式推理技术求解的效率增益也就越大. 取随机数据测试，当 $n=10$ 时，LOGCE 用不到 C 执

行时间的 1/25，LOGCC 用不到 C 执行时间的 1/250，就找到一个近优解(表列情形恰好正是最优解). 当 n=20 时，C 花了 45 分钟仍未找到最优解，而 LOGCE 只用了 40 秒、LOGCC 只用了 4 秒就找到一个近优解.

LogC 已经过较长一段时间的运行和考验，以上列出的比较例子只是若干 AI 典型用例中的两个. LogC 应用于开发大型智能软件系统环境的工作正在进行之中.

§ 3. LogC 环境

与一般的软件工程系统相比，智能软件系统多采用渐增式的原型设计方法. 为支持增量设计，LogC 环境不仅可以使正在设计的程序立即执行，而且可以对程序所进行的推理过程进行跟踪监视，对推理控制策略进行调整，交互式地对程序进行修改和扩充.

LogC 环境按结构模块方式设计. 环境基于 UNIX，总体如图 1 所示.

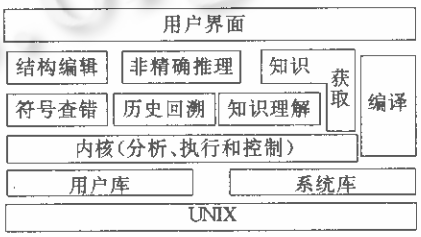

图1

LogC 环境提供有安装工具，允许用户根据自己的需要对环境进行设置和剪裁，从而建立起适合自己配置的用户环境. 内核是 LogC 环境的基本部分，连接系统库可形成一个最小配置的 LogC 执行环境，包括程序装载、程序执行、推理控制、历史回溯、SHELL 命令等部分. 图 1 中的其他各结构块都是可任选结构块.

结论：智能软件系统要求有表示知识并进行推理的能力，为支持这类系统的设计，现有的一些程序设计语言和环境往往不能很好地适应这个要求. LogC 旨在为这类智能软件系统的设计提供一种通用的面向知识推理的语言和环境.

LogC 采用多种程序设计模式，面向过程的计算与面向规则的推理互相交融. 它以 C 作为基语言，象函数一样扩充了规则集. LogC 编程仍然保持着 C 程序设计方便、灵活的特点，从编程风格上不会给用户增加额外的负担. 但 LogC 在实质上已大大超过了 C 而成为一种推理型语言，具有强的启发式推理能力是 LogC 的重要特色，LogC 将一些公共的推理决策寓于环境系统内部之中.

LogC 环境既具有通常一些交互式程序设计环境的特点，又具备自己特有的灵活的知识推理机制. 环境内各工具子系统按结构化模块集成，组装和扩充方便. LogC 语言和环境已实现在 SUN－Station，U－Station，PC386 等机器上，可用于设计各种专家系统、变换系统、决策系统、认知模型、辅助教学等基于知识处理的智能软件系统. 可以期望，LogC 将会作为一个独具特色的新的语言和环境为各种智能软件的系统设计服务.

参考文献

1 D. Bobrow and M. Stefik, The LOOPS Manual, Xerox Corp., Palo Alto, 1983.
2 C. L. Forgy, The OPS83 Report, Tech. Report, CMU－CS－84－133, 1984.
3 B. Hailpern, Multiparadigm Languages, IEEE Software, Vol. 3, No. 1, 1986.
4 C. Ramamoorthy et al., Software Development Support for AI Programs, IEEE Computer, Vol. 20, No. 1, 1987.
5 智能推理语言和环境 LogC 用户手册，中国科学技术大学计算机系软件实验室，1990.
6 智能推理语言和环境技术报告汇编，中国科学技术大学计算机系软件实验室，1990.