中山大學
SUN YAT-SEN UNIVERSITY

本 科 生 毕 业 论 文

题　　目：基于 web 的动物知识专家系统研究与实现

院　　系：软件学院

专　　业：软件工程（计算机应用软件)

学生姓名：莫宇诚

学　　号：11331247

指导教师：伍丽华（高工）

（职 称）

二〇一五 年 四 月

摘　要

专家系统技术经过了几十年的发展，取得了许多成绩。PROLOG 语言作为一种影响力广泛的逻辑编程语言在专家系统中得到广泛应用。web 技术则给一般计算机用户带来了很多的便利。文章简要回顾了专家技术的发展，以完成一个完整的小型的动物知识 web 专家系统 Lucy 为最终目标，详细地了研究了一个专家系统的推理机、人机界面，并最终成功实现。

专家系统常使用 PROLOG 来构建，而 PROLOG 还没有一种通用有效、广泛使用的跨语言通信方法，这不利于专家系统开发有效重用其他编程语言的成果。围绕 PROLOG 实现的推理机和 web 通信上，本文详细地研究了 PROLOG 语言和其他语言的通信方法，分析了其中通用性方法的困难，对过程进行了一般性的分析，提出了被动式查询模型，并设计了一个通用的 PROLOG 接口网络协议，提供了专家系统中纯推理部分和人机交互部分分离的一种方法和尝试。这个协议最后成功被实现。

Lucy 专家系统后端为典型的知识库和推理机，以一个典型动物知识专家系统为原型加以简单修改，并接入了 PROLOG 通用接口协议。推理机部分仅是作为试验地完成了简单知识推理，但并不失一般性。Lucy 成功设计实现了 web 后台和前端。论文讨论了一些开发 web 界面过程中遇到的问题及相关处理、优化、配置的方法，简单模拟了分布式专家系统后端的方法，并讨论了在部署阶段的相关注意事项。Lucy 最终成功实现了 PROLOG 实现的推理系统、web 后台、web 前台的相互通信。

关键词：专家系统 PROLOG 编程语言接口 web 技术 通信协议 被动式查询模型 分布式专家系统

Abstract

After decades of development, expert system technology has made many achievements. PROLOG language, as an instance of logic programming language, is widely used in the expert system development. And the web technology is brought many computer users a lot of user friendly. Reviewing the development of them, to complete a knowledge of animals web expert system, Lucy, with a web user interface as the ultimate goal, this article research a expert system in detail : the reasoning machine, man-machine interface, and eventually finish it successfully.

Expert system often build with PROLOG language. However, PROLOG has no general, effective and widely used method of cross-language communication, this is bad for expert system development effective reuse other programming language. Around reasoning machine write by PROLOG and the communication with web, this article studies in detail communication between PROLOG and other language, analyzes why a general method is difficult. The Passive Query Model is put forward. Designed the General Interface of PROLOG language network protocol, provides a method for communication between the reasoning part and the web back-end.

This system implements a typical expert system, which include a knowledge base and reasoning machine, which are modification from a typical animal expert system prototype. Reasoning machine part is just as a testbed completed the simple knowledge reasoning, but fairly representative.

Lucy also implement the web server. This paper also discussed some problems in the process of the development of web server. In the end, the system successfully let the reasoning system, web back-end, web front-end communicate with each other.

Keywords: Expert System PROLOG Programming Interface web Communicate Protocol Passive Query Model Distributed Expert System

目　录

第一章 前言 6

1.1 背景和意义 6
1.2 论文结构简介 7

第二章 人工智能技术与专家系统 7

2.1 人工智能技术 7
2.2 PROLOG 语言 8
2.3 专家系统技术 9

第三章 LUCY 系统结构简介 10

3.1 LUCY 系统简介 10
3.2 系统设计理念 12
3.3 系统工作流程 13

第四章 PROLOG 的专家系统通用接口研究 14

4.1 通用接口的复杂性 14
4.2 查询 15
4.3 被动式查询 16
4.4 被动式查询与主动式查询的转换 17

第五章 通用接口协议的设计与实现 18

5.1 PROLOG 通用接口协议 18
5.2 协议的通用性 19
5.3 数据编码 20
5.4 编程实现 20

第六章 推理系统的实现 21

6.1 知识库与推理机 21
6.2 知识来源 22
6.3 推理过程 23
6.4 通用接口的接入 25

第七章 WEB 界面实现 26

7.1 使用客户端 26
7.2 服务器后台的实现 26

7.3 服务器前台的实现 26
第八章 分布式专家系统的实现与讨论 27
8.1 架构与流程 27
8.2 分派策略 28
8.3 复合型专家系统 28
第九章 部署与应用 29
9.1 部署 SWI-PROLOG 29
9.2 部署服务器 29
9.3 模拟分布式专家系统后端 30
9.4 设置防火墙 30
第十章 结论 30
致谢 32
参考文献 33
附录 34

第一章　前言

1.1　背景和意义

1965 年英国人费根鲍姆（E.A.Feigenbaum)开创了基于知识的专家系统（Expert System)这一人工智能的新领域[1]。专家系统按其发展过程可分成三个阶段：初创期(1971 年前)，成熟期(1972-1977 年)，发展期(1978 至今）[2~4]。PROLOG (Prog ramming in logic)语言是以一阶谓词逻辑为理论基础的逻辑程序设计语言 ,是人工智能程序设计语言族中应用最为广泛的一种语言[5]。它创建在逻辑学的理论基础之上，常常用于处理涉及目标或关系的问题[6]。

PROLOG 程序基本包括了事实，规则，问题的编程[6]。PROLOG 重要的语义概念还包括量词、原子、变量等[7]。这种逻辑编程范式为建造专家系统中知识库、推理机的编写提供了极大的便利，但其与一般使用的命令编程范式、面向对象编程范式有着许多的差异。

专家知识融合了对问题的理论理解以及大量被经验所证实的启发式问题求解规则。专家系统就是从人类专家那里获取这些知识，然后将其进行形式化编码，使计算机可以应用这些知识来求解类似问题[8]。

一般认为，人机交互界面是专家系统的一部分[9]。另外一方面，专家系统的建造也相对独立地完成，纯专家系统和人家交互界面在一套专家系统中高度融合。PROLOG 语言和其他编程语言之间没有通用的通信方式，专家系统之间也没有通用的通信中间形态。也有了用户越来越希望有一种以用户为中心的通用性专家系统的需要[10]。

本文详细地研究了 PROLOG 语言和其他语言的通信方法，并设计了一个通用的 PROLOG 通用接口协议，提供专家系统中纯推理部分和人机交互部分分离的一种方法和尝试。这种协议也可以适用于各种支持网络的智能代理。

露西（Lucy）是 1974 年在埃塞俄比亚发现的南方古猿阿法种的古人类化石的代称。本文以完成一个完整的小型的动物知识专家系统 Lucy 为最终目标，实现了一个完整的动物知识专家系统，详细地了探索了一个专家系统的方方面面，并验证了设计的通用接口的可用性。

1.2 论文结构简介

本文第一章主要介绍了全文的背景和大致结构。

第二章介绍 Lucy 系统使用到的相关技术。

第三章介绍了 Lucy 系统的总体结构、工作流程和用户界面。

第四、第五章研究了 PROLOG 语言和其他语言的通用通信方法，介绍了 PROLOG 通用接口协议的设计和实现。

第六章是 Lucy 专家系统中，推理机和知识库部分的说明，并介绍了对 PROLOG 通用接口协议的接入和使用。

第七章是 Lucy 专家系统中人机界面的部分。讲述了 web 后台、web 前端，和对通用接口协议的使用。

第八章是 Lucy 修改为适合分布式部署的相关讨论。

第九章是系统的部署、单机模拟分布式专家系统的方法。

第九章是全文的总结。

第二章　人工智能技术与专家系统

2.1 人工智能技术

专家系统属于人工智能的一个分支。美国斯坦福大学著名的人工智能研究中心尼尔逊（Nilson）教授这样定义人工智能“人工智能是关于知识的学科——怎样表示知识以及怎样获得知识并使用知识的学科[11]”，另一名著名的美国大学 MIT 的 Winston 教授认为“人工智能就是研究如何使计算机去做过去只有人才能做的智能的工作[12]”。除此之外，还有很多关于人工智能的定义，至今尚未统一，但这些说法均反映了人工智能学科的基本思想和基本内容，由此可以将人工智能概括为研究人类智能活动的规律，构造具有一定智能行为的人工系统[13]。

人工智能研究途径和方法主要有如下 3 种[9]：

1. 根据人脑的生理结构和工作机理，实现计算机的智能。这种方法基于人脑的生理模型，采取数值计算的方法，从微观上模拟人脑来实现机器智能。

2. 以人脑的心理模型，将问题或知识表示成某种逻辑网络，采用符号推理的方法，从宏观上模拟人脑的思维，以功能模拟、符号推演研究人工智能。

3. 模拟人在控制过程中的智能活动和行为特征，基于感知—行为模型的研究途径和方法。

应该说，专家系统很大程度上是属于第 2 种，也就是符号推理的方法。Lucy 中推理系统采用的方法即是通过建立判断动物的逻辑条件来确定是什么动物。另外一方面，我们看到这些方法不是非此即彼的。本文提出的通用接口方法，实际上也适合被以上三种途径和方法用于跨系统的通信之中。

从 20 世纪 80 年代末开始，专家系统逐步走向多技术、多方法的综合集成与多学科、多领域的综合应用型发展。大型专家系统开发采用了多种人工智能语言、多种知识表示方法、多种推理机制和多种控制策略相组合的方法的方式，并开始运营多种专家系统外壳、专家系统开发工具和专家系统开发环境等[14]。

2.2 PROLOG 语言

人工智能所解决的问题并非一般的数值计算或数据处理问题，人工智能程序更加面对问题、面对逻辑。用常规的过程性程序设计语言进行人工智能程序设计，显得不那么得心应手，于是，面向人工智能的程序设计语言便应运而生，得到广泛的使用。

逻辑性程序设计语言起源于 PROLOG（PROgramming In LOGig）。PROLOG 语言首先由法国马赛大仙的 Colmerauer 和它的研究小组于 1972 年研制完成。在 PROLOG 程序中一般不需要告诉计算机“怎么做”，而只需告诉它“做什么”。PROLOG 语言是以 Horn 子句逻辑为基础的程序设计语言，是目前最具代表性的一种逻辑程序设计语言。

PROLOG 语言只有三种语句，分别表示事实，规矩和问题[9]。

1. 事实（fact)

格式　　<谓词名>(<项表>)

例如：

student(john).

like(mary, music)

2. 规则（rule）

格式　　<谓词名>(<项表>) :- <谓词名>(<项表>){,<谓词名><项表>)}.

例如：

bird(X) :- animal(X), has(X, feather).

3. 问题（question)

格式　　?- <谓词名>(<项表>){,<谓词名>(<项表>)}.

例如：

?- student(john).　%%　真

?- like(mary,X).　%%　X==music

一般而言，问题是程序运行的目标。

可以看出，PROLOG 程序没有一般面向过程或者面向对象语言的许多语法结构：

1. 没有控制流(control flow)。事实上，在编写程序中，规则的多个子句往往用于线性的控制。多个规则实现了分支的功能。而循环则通过递归来实现。

2. 没有函数调用(function call)。事实上，规则并不是函数调用，而是推导规则。而求解一个问题则可以模拟调用一个函数。

另外，PROLOG 程序也有一些普通程序中没有的特点：

1. PROLOG 程序的求解问题过程，是一个语言运行时自动完成的深度优先搜索过程。

2. PROLOG 的谓词（term）数据结构。如 like(mary,X) 是一个数据结构，而且谓词可以嵌套，不存在语义上的化简，如 like(mary, who(X)) 。

PROLOG 语言在求解逻辑推理问题时表现非常优秀，但由于它特别的逻辑编程范式，在与其他编程语言通信时实际上会遇到一定无法完全语义一致的困难，如其谓词类型，并不是任何一种编程语言的内置类型。

2.3 专家系统技术

专家系统的知识有许多表示方法，一般地可以表示为规则和对象。[15]自从 1965 年世界上第一个专家系统 DENDRAL 问世以来，专家系统的技术和应用取得了长足的进步和发展。专家系统亦称专家咨询系统，它是一种能像人类专家一样解决困难、复杂的实际问题的计算机软件系统。这也是 PROLOG 语言构建专家系统的优势所在。

在人工智能早期研究的十多年间，尽管在推理和搜索方面取得不少成就，但这些

技术仍不足以很好地解决复杂的实际问题。专家系统的诞生，使人工智能的研究从以推理为中心转向以知识为中心，为人工智能的研究开辟了新的方向，也使得人工智能有了实际应用的可能。

第三章　Lucy 系统结构简介

本章将简要介绍 Lucy 的系统结构及工作流程，这是一个基于 web 的小型动物知识专家系统。

3.1　Lucy 系统简介

Lucy 使用 PROLOG 完成推理机和知识库的部分，并基于 web 提供人机交互界面。其架构如下：

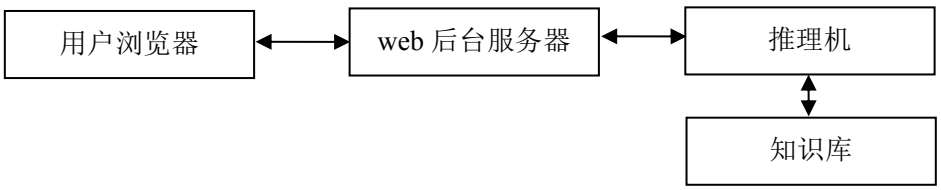

图 3-1　系统架构图

在一次用户请求专家系统服务中，用户使用 web 浏览器访问 web 网站，web 后台服务器与 PROLOG 实现的推理机通信。Lucy 的知识库为静态知识库，推理机来自 PROLOG 本身的机制。

在使用 Lucy 的过程中，用户回答专家系统提出的动物的相关提问，最后推测出用户描述的动物是哪种动物。

图 3-2 Lucy 界面（回答问题）

图 3-3 Lucy 推测出动物名称

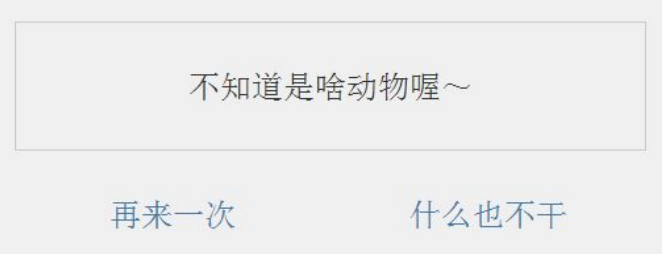

图 3-4 Lucy 无法推测出是什么动物

3.2 系统设计理念

专家系统已经取得了各式各样的应用，但理论上没有大规模的有效革新。笔者希望本文的研究尽可能地对专家系统的相关理论有总结、回顾与新的尝试。总体而言，设计的理念如下：

1. 重视方法的通用性和可适用性。

有效的专家系统有着它的专业性：专家系统是一个具有大量的专门知识与经验的程序系统，它应用人工智能技术和计算机技术，根据某领域一个或多个专家提供的知识和经验，进行推理和判断，模拟人类专家的决策过程，以便解决那些需要人类专家处理的复杂问题。

这种专业性，某种程度上其实专家系统的方法、理论，需要在通用性和专业性之间平衡。过于抽象的设计容易使得专家系统不够通用，而缺少足够的通用性又使得一种方法和理论很难运用在其他地方。

本文的许多讨论围绕通用性进行。可以看出，虽然专家系统的推理机部分较为简单，但是整个系统在基于 web、专家系统等方面的讨论并不失一般性。

2. 保持简单

方法应该尽量的简单，遇到多种方法时，选择简单可行的。子系统之间应该直观和清晰，概念应该容易理解。这要求着达到一种建模和现实要求的平衡。这一点对专家系统而言并不容易，许多专家系统都使用着相似的理论基础，而在高层次的设计，也即是相对面向相关领域的模块时，系统变得混乱和难以解释。

本文提出的通用接口协议也是基于这个原则制定的。它一方面能够足够通用，一

方面又看上去十分简洁，在系统实现过程中，也被证明能够有效、优雅地应对许多复杂的场合。

3. 分而治之

分而治之建立在模块化清晰，抽象良好的前提下。本文实际上大方向就把整个系统分成了 web 前端、web 后台、专家系统后端等部分。而在通用接口协议上，又分成接口协议在 PROLOG 的实现、编程语言接口的实现两部分。

3.3 系统工作流程

用户视角下的软件使用流程如下：

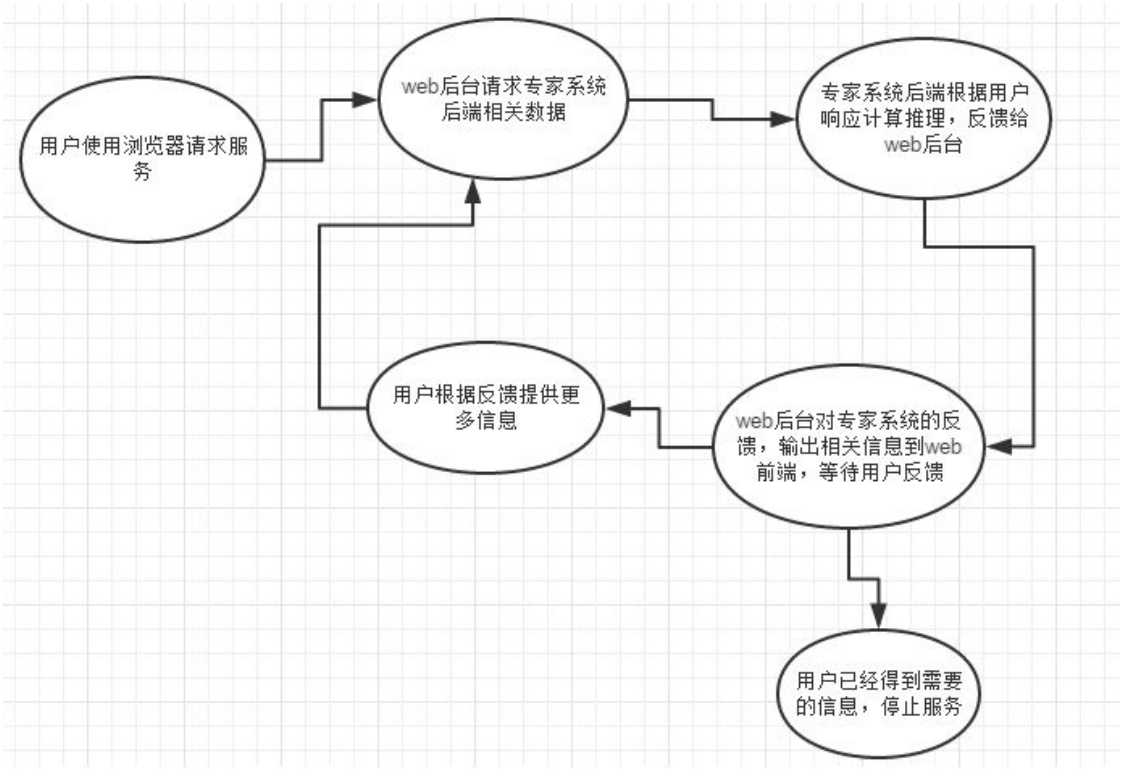

图 3-5 系统工作流程

首先，用户使用浏览器请求服务，web 后台请求专家系统后端相关数据，专家系统后端根据用户响应计算推理，反馈给 web 后台，web 后台对专家系统的反馈，输出相关信息到 web 前端，等待用户反馈，此时用户得到了第一个完整的页面，并需要

作出反应。在 Lucy 中，即是对动物的相关形态进行的提问。

接着，重复再 用户输入 - web 后台 - 专家系统后端 的循环中，用户根据专家系统后端的需要作出反馈，专家系统求解、推理，直到问题得到解决。在 Lucy 中，是更多的追问以确定是哪种动物。

最后，专家系统后端反馈给用户最终的结果。用户得到其所描述的动物的资料。

第四章　PROLOG 的专家系统通用接口研究

4.1 通用接口的复杂性

在本文点 2.2 和 2.3 中我们已经提到 PROLOG 与其他编程语言通信的困难之处。本章节着力于设计一种通用的 PROLOG 与其他编程语言通信的方法。

进一步地，可以发现这其中的复杂性包括但不限于：

1. 跨编程范式带来的复杂性。PROLOG 为逻辑编程范式的编程语言，在与面向过程和面向对象编程语言的通信中，这种通用接口的方法要能够有效适配 PROLOG，且能够在一般面向过程和面向对象的编程语言用使用。

2. 专家系统使用场景的复杂性。我们知道，不同的专家系统用于不同的领域、解决不同的问题，这意味着这种通用接口要有效适用于各种应用场合。

3. 专家系统支持功能的复杂性。不同的专家系统除了一般的推出方案，可能还有提出更多的解，解释一个解等各种功能。

4. 复杂部署环境的复杂性。不同的专家系统可能部署在不同的硬件设备，或者说不同的智能代理上。不同的程序、环境能够良好地进行通信。

5. 复杂编程语言实例复杂性。面向过程和面向对象编程语言也有着不同的编程语言实体，他们的设计也可能不同。

4.2 查询

考虑一个面向过程函数的调用，如：

```
void qsort ( void * base, size_t num, size_t size,
```

int (* comparator) (const void *, const void *));

这样一个调用意味着完成一个过程或者求解一个问题。而这样的一次调用，往往在程序内就实现了程序的执行，不需要再与用户交互。但在一次完整的问题求解，亦即是一次专家系统的完整使用上，往往是多次与用户交互：

1. 用户求解：请告诉我这是什么动物？

2. 专家系统分析：需要知道这个动物是否有羽毛

3. 用户回答：这个动物有羽毛

4. 专家系统分析：需要知道这个动物是否会飞

5. 用户回答：这个动物不会飞

....

N. 专家系统回答：这个动物是企鹅，（出示图片），对吗？

N+1. 用户回答：是的，原来这个东西叫企鹅。

不失一般性地，我们将这样子一次用户和专家系统相互交流，最后得到一个结果的过程，称为一个“查询”。容易发现，专家系统即是这样一个“处理查询”来解决问题的系统。用户和专家系统的一步对话，称为一次“反馈”。专家系统为了产生解而对用户进行的一步对话，称为一次“提示”。

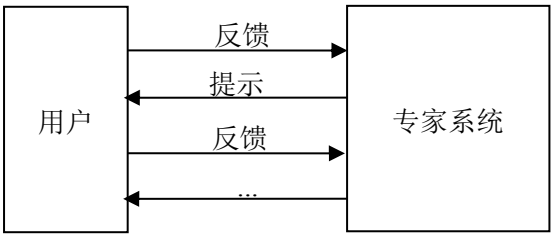

图 4-1 查询的一般过程

而 PROLOG 和编程语言的接口，即是在一次查询中，产生解的过程中提示和反馈在 PROLOG 和编程语言之间的通信方式，以可以采取更好的方式呈现信息给用户。

如专家系统分析“需要知道这个动物是否有羽毛”，用户在其他编程语言上收到这个提示后，其他编程语言使用其完成的 GUI 进行对用户的提问。

容易发现，接口的功能越多，编程的便利性越高，但容易减弱其应对复杂性的能力。而接口的功能越少，编程上就需要做更多的事情，但更容易用于不同的复杂场合。

4.3 被动式查询

进一步地，我们可以把专家系统的查询分为两类：主动式查询和被动式查询。

在主动式查询中，提示仅供用户参考，求解的过程主要靠用户主动提供更多的信息。

1.反馈：这是一只什么动物？

2.提示：目前还无法知道。

3.反馈：它有羽毛。

4.提示：目前还无法知道，可以告诉更多信息吗。

5.反馈：它不会飞。

6....

7.提示：它可能是企鹅。

主动式查询中，用户主动提供信息，专家系统依赖用户提供的信息调整解的范围，直到用户满意。这种经验与大多数的用户界面使用类似。一般的 PROLOG 解释器就一个是主动式查询的过程。

另外一种则是用户一直处于回答专家系统的回答之中：

1.反馈：请告诉我这是什么动物？

2.提示：需要知道这个动物是否有羽毛

3.反馈：这个动物有羽毛

4.提示：需要知道这个动物是否会飞

5.反馈：这个动物不会飞

....

N. 提示：这个动物是企鹅，（出示图片），对吗？

N+1. 反馈：是的，原来这个东西叫企鹅。

在这种查询中，用户一直被动地做出反馈。用户的回答取决于专家系统的提示。

对主动式查询和被动式查询进行简要对比，我们可以发现：

1. 主动式查询更加灵活，被动式查询结构化明显、行为可以预测。主动式查询中无法预知用户的反馈，用户的反馈可能给专家系统带来不同的效果。而被动式查询中，用户的反馈可以预测，专家系统可以枚举反馈的结果来设计专家系统。

2. 主动式查询对用户有着更高的要求，被动式查询用户只需要对提示进行反馈。主动式查询中，用户如果没有有意识地去收敛解的边界，可能解一直无法得到。而被动式查询总可以通过收集需要的反馈，将解收敛成若干个或者证明无解。

3. 主动式查询接近 PROLOG 本身的解释器机制，是一个将反馈转化为提问语句的过程，而被动式查询需要专家系统的设计者大量地去设计求解的路线，工作量更大。

结合实际情况，大多数的专家系统的查询属于被动式查询。事实上，被动式查询在这些地方非常适合专家系统：

1. 结构化的反馈、可以预测的行为有利于保证程序编写的正确性和有效性。

2. 用户只需要对提示进行反馈，可以获得更好的使用体验。

3. 专家系统设计者有更多的空间去完善专家系统的设计。

4.4 被动式查询与主动式查询的转换

被动式查询和主动式查询并不是绝对的，二者其实可以相互转化，这也说明被动式查询的接口的实现具有一定的普遍性。

考虑这样子的谓词

HintAllInput(Input, X)

其中：

1. 如果 Input 能够产生新的提示 Y，将 X 限制为 Y。

2. 否则 X 约束为 fail.

对于一个主动式查询系统 P ，设对于 X 不为 fail 的 Input 的集合为 I 。如果 I 可以枚举，通过设计一个枚举 I 的被动式查询系统 Q，可以实现与 P 一样多的功能。

对于一个被动式查询系统 Q，其提示显然是可以枚举的。那么构造出一个 HintAllInput 谓词，主动式查询系统 P 总是对他的输入响应 HintAllInput ，那么它可以实现与 Q 一样多的功能。

由此可以看出，对于一个专家系统，被动式查询系统对于 I 集合可枚举的情况，总是可以实现。这也意味着，如果 I 集合可枚举，主动查询-被动查询相互切换的查询过程，总可以转换为被动式查询过程。

第五章　通用接口协议的设计与实现

5.1 PROLOG 通用接口协议

Lucy 设计并实现了一种支持被动式查询的通用接口封装协议。定义如下四个指令：

1. START : 表示一次查询开始。
2. ASK(hint) : PROLOG 对用户输出提示 hint。
3. REPLY(reply) : 用户对 PROLOG 输入反馈 reply。
4. FINISH(answer) : PROLOG 对用户输出查询结果 answer。

完整的协议构成如下：

1. 使用 socket 为不同的用户程序和 PROLOG 程序建立连接。
2. 连接建立后，用户程序向 socket 中发送 START 指令。
3. PROLOG 程序收到第一个 START 后，
4. PROLOG 程序发送 ASK 指令。
5. 用户程序根据收到的 ASK 指令收集用户反馈 reply，发送给 PROLOG 程序。

6. 如果 PROLOG 程序已经求出解，转 7。否则转 4。

7. PROLOG 程序发送 FINISH 指令，双方切断连接。

如对 3.2 中查询，过程如下：

- [USER] START
- [PROLOG] ASK(feathers)
- [USER] REPLY(y)
- [PROLOG] ASK(fly)
- [USER] REPLY(n)

...

- [PROLOG] ASK(is(penguin))
- [USER] REPLY(y)
- [PROLOG] FINISH(penguin)

容易发现， START 指令发送后，在被动式查询系统中，总是 ASK - REPLY 的循环。这使得 socket 的编写非常容易，只需要一方监听，一方写回即可。

5.2 协议的通用性

考虑使用 socket 建立连接，并且使用字符串将指令的参数编码。这使得在支持 socket 的 PROLOG 运行时环境都可以使用本协议。而支持 socket 的编程语言，都可以使用本协议与 PROLOG 通信。此外，由于本协议不依赖于其他的环境和问题。这有效解决了 3.1 中的 1、4、5 提到的复杂性问题。

协议建立在被动式查询的基础上，不同的使用场景和不同的功能，只要可以整理为被动式查询，也总可以通过本协议来解决。这一定程度上解决了 3.1 中的 2、3。

协议没有对 socket 中的数据封装的说明，也没有对具体的 PROLOG 数据封装做出了规定，这是在通用性和编程灵活性上的平衡。

除了可以作为跨语言、跨程序间的接口通信协议，本程序也可以作为 PROLOG 程序间通信的协议，此时可以将多个智能代理实体连接起来。不同功能的专家系统也可以被同一套方法进行编程。

通过本协议，PROLOG 程序可以实现与其前端的分离。专家系统的用户界面可以相对独立地离开 PROLOG 程序的推理部分。

与外壳系统相比，本协议没有提供编程语言间通信的其他工具，但显然具有更高的灵活性。

5.3 数据编码

对于指令中的项如何进行编码解码，有许多可能的方法。

1.不包装为数据结构，直接在指令中传输字符串。这种方法的优点是编码容易，直观简单，但不利于不同程序的分析和提取信息。

2.根据 PROLOG 的语义封装为相应的数据结构。在一般的编程语言中没有谓词这种数据类型，这意味着需要再封装一个 Term 型的数据结构。这种方法的有点是 PROLOG 和其他编程语言的通信可以高效地维持语义的完整，但非内置的数据结构可能会带来编码的麻烦。

3.在 PROLOG 端编码、解码为 Term ，在应用程序端作为字符串处理。这样为了了 PROLOG 处理时的简洁，而编程语言端需要自行解析数据。

Lucy 使用了方法 3。在 PROLOG 端读写时都是 Term，而在 python 端读写只是得到字符串形式的 term 。

5.4 编程实现

一个需要使用本协议与 PROLOG 程序进行通信的编程语言程序，实际上需要编写一套对应的读写 socket 的客户端协议。一般的，面向对象的语言中可以编写成如下的一个类：

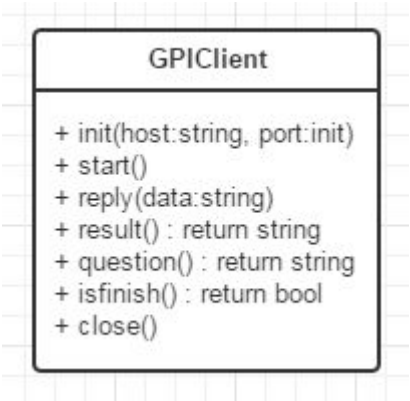

图 5-1 通用 PROLOG 接口类图

init 构造函数指定其要连接的 PROLOG 程序 socket 地址。start 函数发送 START 指令。reply 函数发送反馈，其中 data 是可以被 PROLOG 解码为 term 的字符串。result 函数返回本次查询的结果，如果已经有了答案。question 返回最后一个提示。isfinish 用来判断是否已经结束本次查询。close 用来强制关闭此次查询。

面向过程的编程语言也可以采取类似的封装。

而在 PROLOG 中，需要实现如下的规则：

server(Port, Goal)	监听 Port 端口，每个客户端使用 Goal 谓词来启动查询
session_start	当前查询开始时要被调用，用来检查和初始化
session_ask(Q, R)	在当前查询中，输出提示 Q，将 R 限定为用户反馈
session_reply(A)	当前查询返回 A 为最后结果

第六章　推理系统的实现

6.1 知识库与推理机

Lucy 的专家系统后端部分在一个使用 PROLOG 语言的简单的动物知识专家系统的基础上修改。不失一般性地，其展示了一个完整的被动式查询专家系统的流程。Lucy 使用的是知识为人工获取得到，知识库的内容以 PROLOG 代码的方式保存在程序中。

功能上，支持六种动物的猜测。用户回答问题，专家系统自动分析为哪种动物，并进一步追问。如果不在知识库内，则输出不知道为哪种动物。

推理机的部分即为 PROLOG 求解的过程。

6.2 知识来源

与决策树等方法相比，直接 PROLOG 编程实现的专家有则许多的缺点：

1. 不能直接应对大数据、大规模下的建模，需要调整算法模型。
2. 性能较差。
3. 可能存在冗余提问。

另外一方面，直接 PROLOG 编程也有着其自身的优点：

1. 能够更加灵活地面对非结构化的提问。
2. 更加细致的交互方式。
3. 小型系统开发成本更低。

直接 PROLOG 编程实现的程序中，代码（code）即是数据（data）。也有新的方法（如基于框架的方法），可以使得 PROLOG 适应于更多的场合。Lucy 延续直接 PROLOG 编程实现的方法。

Lucy 的知识来自于一个经典的专家系统例子，整理后如下图。

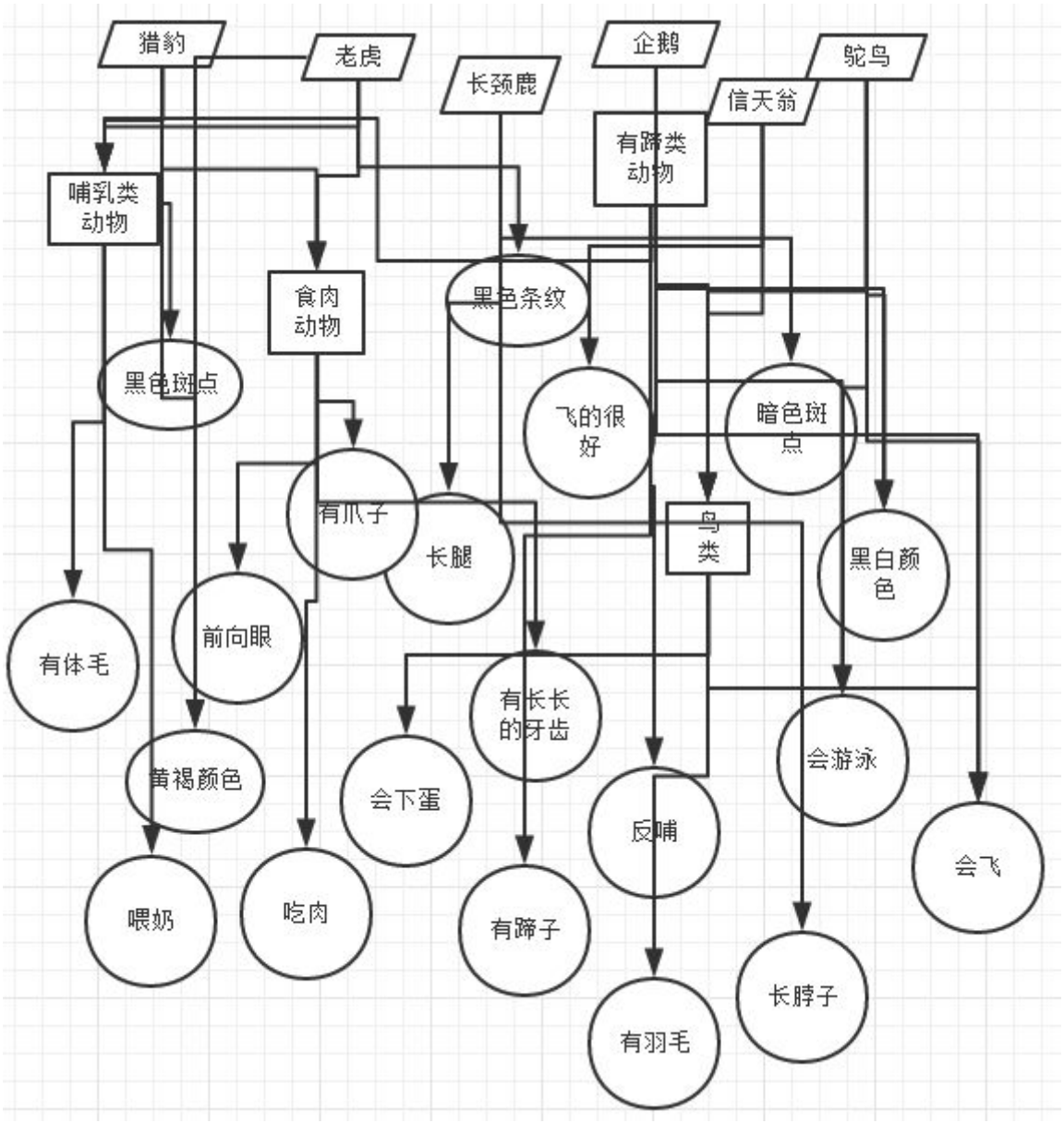

图 6-1　Lucy 包括的动物知识

由于在这个专家系统中知识是相对结构化的，使用决策树也可以取得相对好的解决。但考虑增加根据动物名称查询动物知识、非结构化提问等情况，使用 PROLOG 仍有较好的可扩展性等优点。

6.3　推理过程

一段 PROLOG 代码如下：

```
animal_is(cheetah) :-
    it_is(mammal),
    it_is(carnivore),
    positive(has, tawny_color),
    positive(has, black_spots).
```

其描述了判断为猎豹(animal_is(cheetah))的规则：哺乳类动物 it_is(mammal) ，食肉动物 it_is(carnivore) ， 有着黄色的颜色 positive(has, tawny_color) ，有着黑色的斑点 positive(has, black_spots) 。PROLOG 求解的原理已经超出本文范畴，但可以看出求解的过程，本身就是一个专家系统的推理机部分。

求解某种动物，实际上是定义了各种动物的判断规则 animal_is(cheetah) 、animal_is(tiger) 、 animal_is(giraffe) ... 然后通过求解 animal_is(X) 中 X 约束为何值时可以为真而求出这种动物，否则反馈为未知。

而 animal_is 则根据动物本身的定义进行定义，这其中用到了 it_is 和 positive 、negative 谓词。it_is 是为了编写方便，而定义的一些常见属性、类别，如 it_is(bird) 、it_is(mammal) 、 it_is(carnivore) 。

negative 和 positive 则根据需要处理用户输入相关。如果某个属性已经询问过用户， 例如 positive(has, long_neck) 用户已经回答过这种动物有一个长脖子，那么就 positive(long_neck) 为真，否则需要用户输入，也即是反馈给 web 后台再由浏览器前端等待用户的回答。

经过一系列的交互后，最后找出约束值 X 或者判断为无解。

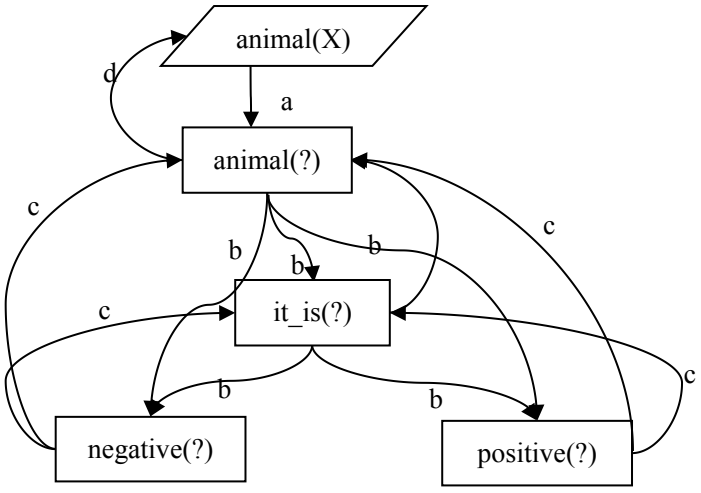

图 6-2 推理机查询过程

如，系统一开始提问是否有体毛，回答有。接着提问是否吃肉，回答不是。接着提问有没有尖尖的牙齿，回答无。接着问是否有哺乳，回答有。接着问是否有蹄子，回答有。接着问是否有长长的脖子，回答有。接着问有没有长长的腿，回答有。接着问是否有暗色的斑点，回答有。专家系统推测出这个动物是长颈鹿。

又如，提问是否有体毛，回答无。接着提问是否哺乳，回答不是。接着问是否有羽毛，回答有。接着问是否会飞，回答不会。接着问是否有长长的脖子和长长的腿，回答有。接着问是否是黑白色的外表，回答是，专家系统推测出是鸵鸟。

如果专家系统无法推测，可以返回为不知道。如一开始回答体毛为没有，不哺乳，没有羽毛，不会飞。专家系统即会表示不知道是什么动物。

6.4 通用接口的接入

根据 4.4 中的规则：

1. 导入 server 函数，并以 server(6554, bird_system) 为程序运行目标。其中 6554 为监听端口，bird_system 为服务谓词。

2. 在 bird_system 中使用 session_start 子句，相对应的分支为正常的客户端连接。

3. 在需要提示以获得用户反馈的地方，使用 session_ask(ask_it(X, Y), Reply) 给客户端发送提示 ask_it(X, Y) ，并限定 Reply 为用户的反馈。

4. 在需要使用动态数据库的地方使用 session_asserta, session_data 谓词，他们是 assert 谓词对单个连接的版本。

5. 在可以确定答案的地方，使用 session_reply(may_be(X)) 提示最后的结果。

使用这套方法，对于原来的命令行人机界面，修改共 6 处，非常方便地接入了通用接口协议。

第七章　web 界面实现

7.1 使用客户端

根据 4.4 中的客户端类图，实现了 python 版本的类 GPIClient ，支持相应的方法。这里用了一个小技巧，在送出本次反馈后，马上读取下次的提示。

7.2 服务器后台的实现

后台使用 python flask 框架完成。每次用户开始一次查询，每个查询 PROLOG 和 web 后台都只维持着一个连接，不能断开，但 HTTP 在请求完数据后即断开。对此：

1. 使用 websocket 等新 HTML5 技术，但其在浏览器兼容性上并不佳。

2. 在服务器后台为每个用户的查询维持一个 GPIClient 实例，每个连接根据 cookie 查到对应的实例。过一段时间都没有活动的实例程序去清除掉。

Lucy 使用了 2 的方法。

7.3 服务器前台的实现

解析 PROLOG 产生的 term 字符串，实际上在服务器前台完成。为此，实现了三个重要的函数：

has_word(word) : 该 term 字符串是否包括单词 word

select(word_list) : 该 term 字符串如果包含 word_list 中的某一项，返回之。否则返回 null

handler(term_string) : 根据收到的 term_string 字符串，调整 web 界面收集用户反馈。这里利用了 term_string 与用户界面是一一对应的。

通过 ajax，浏览器前端抓取 web 后台传过来的数据 term_string ，并通过 has_word 和 select 将其“翻译”为一个友好的界面。在用户在界面上做出反馈后，通过 ajax 提交用户的反馈，并再次抓取新的 term_string 。

第八章　分布式专家系统的实现与讨论

8.1 架构与流程

考虑目前为止完成的系统中，实际上单个 web 后台服务器与单个专家系统后端通信。但由于通用接口协议的工作方式，修改为分布式的专家系统并不困难。

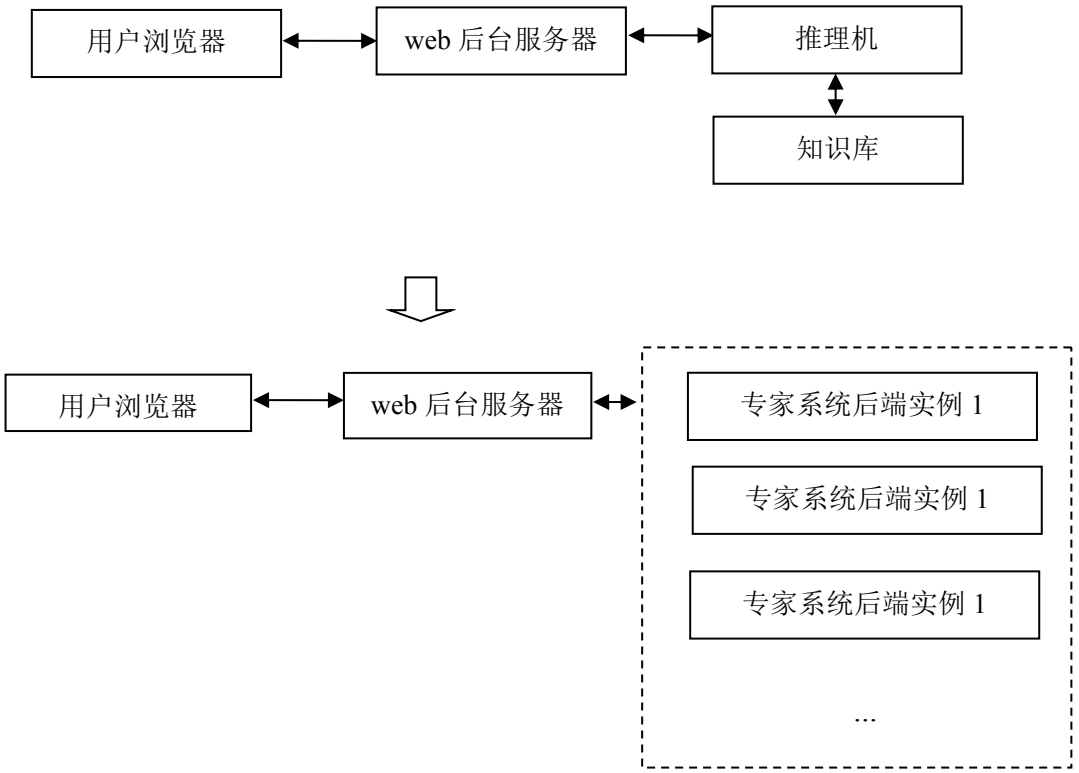

图 8-1 分布式专家系统架构

考虑在原来 web 后台与专家系统后端通信的部分，只需要将这部分动态地分派到某个专家系统后台实例即可。修改后的工作流程大致如下：

1. 用户使用浏览器访问。
2. web 后台根据用户反应，决定要分派到的专家系统后端。
3. web 后台与专家系统后端通信，取得数据。
4. web 后台反馈到用户浏览器。

这样子可以有效地把性能负担分散到多个机器上。这样做可以实现专家系统后端的水平扩展，当性能不足的时候，简单地增加服务器数量即可。而在非单一专家系统后端的场合，只要支持 socket 的场合都可以简单地通过通用接口协议进行通信。

在 Lucy 中，我们使用单机模拟的方式，实现了一个单一 web 后台与 5 个专家系统后端实例的场景，工作良好。

8.2 分派策略

在 web 后端分派到专家系统后端的过程中，分派过程有许多种策略：

1. 根据用户信息分派。例如，根据用户名的首写字母进行分派，根据用户 ip 进行分派。

2. 根据服务器负载进行分派。这样可以充分发挥各个后端机器的性能，但是维持服务器负载信息也需要成本。

3. 随机分派。每次请求随机产生一个随机数，然后分派到相应的后端实例中。优点是操作简便，但可能不利于 cache 和用户不同请求有关联的通信。

可以发现随机分派的性能其实并不差，而且表现相对稳定。

当单点专家系统后端故障后，整个系统也不会完全地崩溃。

另一方面，为了充分发挥现代机器的优势，也可以一个机器部署多个专家系统后端实例进程，使用不同的端口。

```
1  localhost:8001
2  localhost:8002
3  localhost:8003
4  localhost:8004
5  localhost:8005
```

图 8-2　iplist.txt 文件　(配置 PORLOG 专家系统后端网络地址）

```
def choice_client(iplist) :
    host, port = random.choice(iplist).split(':')
    print 'SELECT>> %s:%s' % (host, port)
    return GPIClient(host, int(port))
```

图 8-4 随机分派策略的核心代码（调试用）

在 Lucy 中，我们在 web 机本地部署了 5 个实例，web 后台随机分派到其中一个实例中请求相关查询服务。

8.3 复合型专家系统

除了 Lucy 实现的单 web 后台分派到多个专家系统后端，通用接口协议也可以支持更多复杂的情形：

1. 多 web 后台分派到多个专家系统后端。
2. 多个复杂的代理（Agent）到专家系统后端。
3. web 后台服务、复杂代理（Agent）到专家系统后端。
4. 专家系统后端相互之间的通信。
5. 多种编程语言实例到专家系统后端。

由于使用 socket 进行通信，这些场景并没有带来通信协议、通信接口的本质改变，只要能支持 socket 的场合都能够直接使用。

第九章　部署与应用

9.1 部署 swi-PROLOG

系统使用 swi-PROLOG。支持 linux 平台。大部分都直接使用发行版的包管理工具安装即可。在部署好 swi-PROLOG 后，即可开启 PROLOG 服务进程：

```
swipl -s lucy.pl -g run -- localhost 6554
```

一般的编程语言此时即可通过 6554 端口向专家系统进行查询。需要修改监听端口修改 lucy.pl 中的参数即可。

9.2 部署服务器

在安装好相关需要的包后，开启 python web 服务器：

```
python app.py
```

此时已经可以通过 5000 端口获得服务。需要修改端口修改 app.py 中相关参数即可。

9.3 模拟分布式专家系统后端

样例程序如 app2.py 。其会读取 iplist.txt 。 iplist.txt 是专家系统后端的地址清单，每行一个。web 后端会根据地址清单随机选择一个地址进行连接。如何在多个机器上部署专家系统后端实例已经超出本文讨论范畴。但可以参考 start_all_server.sh 程序在不同的机器上进行部署。在 Lucy 的实现中，这个脚本会读取 iplist.txt 然后在本地开启这些后端。

```
./start_all_server.sh      # 开启所有专家系统后端实例
python app2.py             # 开启 web 后端服务器
```

9.4 设置防火墙

基于最小权限的考虑，如果不需要，不必开发 6554 端口给外部的程序和连接。为此，可以设置防火墙将访问 6554 端口的包扔掉。

在多机器组成的网络中进行部署时，则需要开放相关的防火墙端口，制定合理的安全策略。

第十章　结论

专家系统的发展取得了许多的成就，应用广泛。Lucy 专家系统围绕动物种类推测，成功设计并实现了一个基于 web 的专家系统。

围绕 web 和 PROLOG 的通信接口，Lucy 通过被动式查询模型完成建模，设计并实现了 PROLOG 通用接口协议，通过 socket 通信的方法解决 web server 和 PROLOG 通信，并具有很好的通用性。

通过人工获取的动物知识，完成了 PROLOG 实现的专家系统后端，实现了专家系统的推理机和知识库。通过设计的 PROLOG 通用接口协议，最后成功完成了一个 python 实现的 web 后台，并实现了专家系统后端和 web 的通信，工作良好。

简单地修改 web 后台，使用单个 web server 和多个专家系统后端通信的方法，模拟了分布式专家系统的实现，证明了通用接口协议的可行性，也证明了 Lucy 的扩展性和灵活性。并讨论了 Lucy 部署的相关问题。

本文围绕 PROLOG 通用接口协议的设计和实现，重点解决了 PROLOG 与 web 通信接口问题，最终成功实现了一个功能典型、完整、可行的基于 web 的动物知识专家系统。

致谢

感谢我的导师伍丽华老师对我的悉心指导！
感谢我的家人一直支持我。

参考文献

[1] 吴泉源，刘江宁，人工智能与专家系统，国防科技大学出版社,1995

[2] 蔡自兴，约翰▪德尔金，龚涛，高级专家系统：原理、设计及应用，科学出版社,2005

[3] 敖志刚，人工智能与专家系统导论，中国科技大学出版社，2002

[4] 王永庆，人工智能原理与方法，西安交通大学出版社，1998

[5] 白晓虹，王世勤，PROLOG 语言特点综述.，延安大学学报：自然科学版 1，23-26，1999

[6] W.F.Clocksin，C.S.Mellish，Programming in PROLOG Fifth Edition，2003

[7] Blackburn, Patrick, Bos, Johan,Striegnitz, Kristina， Learn PROLOG Now!,College Publications,2006

[8] George F. Luger, 郭祖茂等译，人工智能：复杂问题求解的结构和策略（原书第 6 版）, 机械工业出版社, 2009

[9] 廉师友，人工智能技术导论(第三版)，西安电子科技大学出版社，2007

[10] 杨兴，朱大奇等，智能故障诊断专家系统开发平台研制，控制工程（增刊），12(7)， 180-182，2005

[11] 贾同兴，人工智能与情报检索，北京图书馆出版社，15-103，1995

[12] 胡 勤，人工智能概述，电脑知识与技术，（13）：3507-3509，2010

[13] 许万增，王行刚等，人工智能对人类社会的影响，科学出版社，21-73，1996

[14] 尚福华，人工智能及其应用，石油工业出版社，石油工业出版社，2005

[15] Giarratona,J. 等著，专家系统原理与编程：第 3 版，机械工业出版社，2002

附录

1. PROLOG 通用接口协议封装代码

```
:- module(pqserver, [
            session_asserta/1,
            session_assertz/1,
            session_retract/1,
            session_retractall/1,
            session_data/1,
            session_start/0,
            session_ask/2,
            session_reply/1,
            server/2
         ]).

:- use_module(library(streampool)).
:- use_module(library(readutil)).

:- set_prolog_flag(report_error,true).
:- set_prolog_flag(unknown,error).

:- meta_predicate
     server(+, :).

:- dynamic
     server_goal/0,
     server_setting/1,   % Name(Value)
     server_session_data/2.     % ThreadID, Data
```

```
session_asserta(Data) :-
    % 取得线程 id，将其通过 server_session_data
    % 谓词结合装入动态数据库，下同
    thread_self(Id),
    asserta(server_session_data(Id, Data)).

session_assertz(Data) :-
    thread_self(Id),
    asserta(server_session_data(Id, Data)).

session_retract(Data) :-
    thread_self(Id),
    retract(server_session_data(Id, Data)).

session_retractall(Data) :-
    thread_self(Id),
    retractall(server_session_data(Id, Data)).

session_data(Data) :-
    thread_self(Id),
    server_session_data(Id, Data).

session_instream(In) :-
    thread_self(Id),
    server_session_data(Id, instream(In)).

session_outstream(Out) :-
    thread_self(Id),
```

```
    server_session_data(Id, outstream(Out)).

session_ask(Q, R) :-
    % 实现 ask 指令，直接写 socket 然后读
    session_outstream(Out),
    write(Out, Q),
    write(Out, '\n'),
    flush_output(Out),
    session_instream(In),
    read_line_to_codes(In, Codes),
    read_term_from_codes(Codes, R, []).

session_start :-
    % start 指令，连接建立后使用 hello 检验
    session_instream(In),
    read_line_to_codes(In, Codes),
    read_term_from_codes(Codes, Atom, []),
    session_start_check(Atom).

session_start_check(Atom) :-
    Atom == hello, !.

session_start_check(Atom) :-
    throw(error(failed(noSessionStartFlag, Atom), _)).

session_reply(A) :-
    % 实现 reply 指令
    session_outstream(Out),
    write(Out, reply(A)),
```

```
    write(Out, '\n'),
    flush_output(Out).

clear_session_data :-
    thread_self(Id),
    retractall(server_session_data(Id, _)).

server(Port, Goal) :-
        asserta(server_goal(Goal)),
        tcp_socket(Socket),
        tcp_bind(Socket, Port),
        tcp_listen(Socket, 5),
        tcp_open_socket(Socket, AcceptFd, _),
        dispath(AcceptFd).

dispath(AcceptFd) :-
    % 使用新线程处理新的连接
    tcp_accept(AcceptFd, Slave, _),
    tcp_open_socket(Slave, In, Out),
    thread_create(client(In, Out), _,
                [ datached(true)
                ]),
    dispath(AcceptFd).

client(In, Out) :-
    session_asserta(instream(In)),
    session_asserta(outstream(Out)),
    server_goal(Goal),
    catch(Goal, E,
```

```
        ( print_message(error, E),
          true
        )),
    close(In),
    close(Out),
    clear_session_data.
```

2. Python 封装客户端

```
import socket
import random

class GPIClient :
    '''通用 PROLOG 接口协议客户端'''

    buf_size = 1024

    def __init__(self, host='localhost', port=6554) :
        self._s = None
        self._host = (host, port)
        self._iswaiting = None
        self._lastdata = ''       # 最新一次提示的数据保存在 _lastdata
        self._reply = None            # 最终结果保存在 _reply 里
        self._isclose = False

    def __str__(self) :
        return '<GPIClient %s:%d>' % self._host
```

```
def start(self):
    # start 指令的实现
    if self._iswaiting != None :
        raise ValueError
    self._s = socket.socket(socket.AF_INET,
                            socket.SOCK_STREAM)
    self._s.connect(self._host)
    self._iswaiting = False
    self._buf = []
    return self.reply('hello')

def reply(self, data) :
    # reply 指令的实现
    if self._iswaiting != False :
        raise ValueError
    self._iswaiting = True
    try :
        self._s.sendall(data + '\n')
        self._lastdata = self._recv()
    finally :
        self._iswaiting = False
        if self._lastdata[:6] == 'reply(' :
            self._reply = self._lastdata
            self.close()
            return None
        else :
            return self._lastdata

def result(self) :
```

```
    '''取得最终结果'''
    if not self._reply :
        raise ValueError("This query has not finish yet.")
    return self._reply

def _recv(self) :
    try :
        sp = self._buf.index('\n')
    except ValueError:
        while True :
            d = self._s.recv(1024)
            if d :
                self._buf.extend(d)
                try :
                    sp = self._buf.index('\n')
                except ValueError :
                    pass
                else :
                    break
            else :
                self.close()
                raise ValueError("Socket has been close")
    line = ''.join(self._buf[:sp])
    self._buf[:sp+1] = []
    return line

def question(self) :
    return self._lastdata
```

```
    def isfinish(self):
        return self._isclose

    def close(self):
        self._isclose = True
        self._s.close()

def choice_client(iplist) :
    '''根据 iplist 列表随机选择一个地址建立连接'''
    host, port = random.choice(iplist).split(':')
    print 'SELECT>> %s:%s' % (host, port)
    return GPIClient(host, int(port))

if __name__ == '__main__' :
    c = GPIClient()
    c.start()
    while not c.isfinish() :
        print c.question()
        c.reply(raw_input().strip())
    print c.result()
```

附表一、毕业论文开题报告

论文（设计）题目：

web 技术在今天取得巨大的发展，论文通过结合 web 和专家系统知识，实现一个简单的动物知识专家系统，并基于 HTTP 协议实现一种有效、通用的在 web 和 PROLOG 编程中通信的方法。

此系统由两部分组成：1. PROLOG 完成的推理机和知识库 2. web 界面提供的人机交互界面。用 PROLOG 完成对知识和推理编程，web 界面收集用户输入的动物特征，通过 PROLOG 完成的核心推理机制推理出相应的动物资料，最后反馈到 web 给用户。

PROLOG（Programming in Logic 的缩写）是一种逻辑编程语言。它创建在逻辑学的理论基础之上， 最初被运用于自然语言等研究领域。专家系统则是早期人工智能的一个重要分支，它可以看作是一类具有专门知识和经验的计算机智能程序系统，一般采用人工智能中的知识表示和知识推理技术来模拟通常由领域专家才能解决的复杂问题。一般来说，专家系统=知识库+推理机。

在接口的部分，通过 远程过程调用（Remote Procedure Call） 的方式，基于 HTTP 设计并实现 web 和 PROLOG 程序之间的通信协议，并封装为函数库。这种方法使得 web 其实可以对多个 PROLOG 查询问答服务器进行通信，也可以在单机上完成多种编程语言对 PROLOG 的调用。

论文主题分成如下几部份：
1. 专家系统的知识库和推理机的完成。收集动物的知识，实现一个结构化、易于分析推理、不失通用性的知识库。并基于知识库和动物知识建立推理机制，根据用户的输入和知识库推理出相应的可能的动物物种。
2. 基于 HTTP 协议设计 web 和 PROLOG 之间的通信协议。包括：
a) 查询协议。规定如何通过 HTTP 查询 PROLOG。
b) 响应协议。规定如何响应一个对 PROLOG 的查询。
3. 实现协议并封装为通用的函数库。使用函数库应该对 HTTP 协议透明。
4. 整个系统的完成。

进度安排：
2014.1 月 完成知识库和推理表示的设计，收集知识
2014.12 月 完成可用的 PROLOG 编程，设计查询和响应协议
2015.1 月 实现协议并封装为函数库
2015.2 月 完成 web 和整个系统

学生签名： 年 月 日
（专业、学号、姓名）

指导教师意见：

1、同意开题（ ） 2、修改后开题（ ） 3、重新开题（ ）

指导教师签名：　　年　　月　　日

附表二、毕业论文过程检查情况记录表

指导教师分阶段检查论文的进展情况（要求过程检查记录不少于 3 次）：

第 1 次检查

学生总结：

阅读了专家系统相关书籍、相关文献，并有了一定程度的理解。提出了系统的大概架构设想。

指导教师意见：

继续完善系统设计，着重探讨人机界面交互相关的问题。

第 2 次检查

学生总结：

提出了完整的系统架构，决定使用基于网络通信的方法实现 web 后台和 PROLOG 后端的通信，并作出了大致的设想。

指导教师意见：

完善论文的理论部分，继续完善通信协议方面的设想，阅读更多的文献。

第 3 次检查

学生总结：

完成了系统的雏形，并可以运行。完成了 web 前端、web 后台、通信协议、PROLOG 后端的实现。

指导教师意见：

完善系统 demo，完善第三、四、七章，增加论文系统的设计说明。

第 4 次检查

学生总结：

尝试基于目前的进度增加了分布式专家系统后端的实现，完善全文。

指导教师意见：

完善论文，尤其注意格式、图片、文献等细节。

<table>
<tr><td colspan="2">学生签名：　　年　月　日

指导教师签名：　　年　月　日</td></tr>
<tr><td>总体完成情况</td><td>指导教师意见：

该生在整个论文写作过程中，主动学习并开展研究，积极认真，按时按要求独立完成论文的各项工作，论文的结构完整，内容充实，主要观点突出，逻辑关系清楚，达到本科毕业论文的要求。

1、按计划完成，完成情况优（ ）
2、按计划完成，完成情况良（ ）
3、基本按计划完成，完成情况合格（ ）
4、完成情况不合格（ ）

指导教师签名：　　年　月　日</td></tr>
<tr><td></td><td></td></tr>
</table>

附表三、毕业论文答辩情况

答辩人		专 业	
论文（设计）题目			
答辩小组成员			

答辩记录：

记录人签名：　　年　　月　　日

学术诚信声明

本人所呈交的毕业论文，是在导师的指导下，独立进行研究工作所取得的成果，所有数据、图片资料均真实可靠。除文中已经注明引用的内容外，本论文不包含任何其他人或集体已经发表或撰写过的作品或成果。对本论文的研究作出重要贡献的个人和集体，均已在文中以明确的方式标明。本毕业论文的知识产权归属于培养单位。本人完全意识到本声明的法律结果由本人承担。

本人签名：　　日期：

毕业论文成绩评定记录

指导教师评语：   成绩评定： 指导教师签名：　　年　　月　　日
答辩小组或专业负责人意见：   成绩评定： 签名（章）：　　年　　月　　日
院系负责人意见：   成绩评定： 签名（章）：　　年　　月　　日